NTIA Report 90-266

Transmission Monitor and Control Software Reference Manual

Richard N. Statz

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary

Janice Obuchowski, Assistant Secretary
for Communications and Information

June 1990

DISCLAIMER

CONTENTS

CONTENTS (cont.)

LIST OF FIGURES

LIST OF FIGURES (cont.)

ACRONYMS

DEB - Digital European Backbone

The American military communications system in Europe is in the process of upgrading from analog to digital technology. The new digital system is called the Digital European Backbone. The TRAMCON software system has been developed in conjunction with the DEB, but is not limited to monitoring only the DEB.

DS - Distributed Systems

Distributed Systems is a software package supplied by the computer vendor (Hewlett Packard). This software manages a network (IPC, defined below) that connects the TRAMCON computers.

EMA - Extended Memory Area

The Hewlett Packard 1000 minicomputer is a 16-bit word size machine that limits addressing to 32767 (2^{16}-1). This is inadequate for today's applications. To allow a given program to access more than 32767 words of data stored in central memory, a feature called EMA was added to the RTE6/VM operating system. This EMA facility uses two-word addressing to allow application programs, such as the TRAMCON programs, to access all the memory that may be installed on the machine. The TRAMCON software maintains all its static configuration data and real time data in a portion of EMA memory. Another feature of this EMA that TRAMCON takes full advantage of, is the ability of separate programs to share access to the same data stored in EMA.

HP - Hewlett Packard Corporation

The TRAMCON application software is based upon an HP-1000 minicomputer.

HP-1000 - Hewlett Packard 1000, Series F minicomputer

This minicomputer is the hardware foundation of the TRAMCON On-Line system. In this document the HP-1000 is also referred to as a TRAMCON Master Terminal, or TMT.

IPC - InterProcessor Communication

The TRAMCON computers are connected to one another via a synchronous 2400-bps HDLC communications link. The network is defined so that any given TRAMCON computer can communicate with any other TRAMCON computer, although each computer is directly connected to no more than two other computers. The DS software described above is used to perform the communications over this IPC network.

ITS - Institute for Telecommunication Sciences

The Institute, within the U.S. Department of Commerce, developed the TRAMCON computer system and this document which describes the TRAMCON system. ITS is the research and engineering unit of the National Telecommunications and Information Administration (NTIA) and is located in Boulder, Colorado.

I/O - Input/Output

The communication between any digital computer and the outside world (e.g., peripheral devices, data lines) is referred to as I/O.

LU - Logical Unit

The TRAMCON software refers to the physical I/O channels on the HP-1000 minicomputer by using Logical Unit numbers. These LU numbers are associated with default I/O channels at operating system generation. They can be reassigned, either by the TRAMCON operator or programmatically, to other I/O channels anytime while the software is running. This flexibility allows programs to communicate with different devices without being recompiled and reloaded.

RTE-6/VM - Real Time Executive operating system

The RTE-6/VM operating system, version A.85, was used to implement the TRAMCON application software on the HP-1000 minicomputer.

TMT - TRAMCON Master Terminal

"TRAMCON Master Terminal" or "TMT" is the name given to the HP-1000 minicomputer used to implement TRAMCON. Each TMT consists of an HP-1000 F-series minicomputer with 1.5 Mbytes of central memory, a 65-Mbyte Winchester technology disc drive (model 7912) with integral streaming tape drive, and from one to four color display terminals.

TRAMCON - TRAnsmission Monitor and CONtrol

This manual describes the TRAMCON applications software developed by ITS for the U.S. Air Force Electronic Systems Division (ESD) and serves as a guide for the maintenance of this software. The TRAMCON software monitors any communications equipment down to the T1 level. The TRAMCON software also allows the reconfiguring of this transmission equipment via remote controlled relays.

GLOSSARY

CATEGORY - The TRAMCON data collection devices (remote units) can report information about the SITE equipment (e.g., battery voltage, door intruder alarm) and information about up to three sets of communications equipment. These sets of equipment are referred to as CATEGORIES.

LINKEND - A LINKEND is defined as one CATEGORY or one set of communication equipment that is located at a SITE and transmits/receives in a given direction. A communications LINK consists of two LINKENDs communicating with each other.

LOGICAL REMOTE UNIT - A LOGICAL REMOTE UNIT is composed of one or more PHYSICAL remote units (see definition below). When a site contains more sets of communications equipment than can be monitored by one physical remote unit, several units must be used to monitor the entire site. Nevertheless, the software presents all the information for any given site to the operator as if it were monitored by one remote unit.

MASTER - Each TRAMCON minicomputer is referred to as a TRAMCON master. The master's identification is derived from the site where it is located. Each TRAMCON master polls a set of data collection devices (Remote Units), receives responses from these devices, processes these responses, and presents the status of the transmission and facility equipment to the TRAMCON operator.

MONITOR - Each TRAMCON master computer can operate in one of two modes, either MONITOR or POLLER, for each TRAMCON segment (see definition below) that it monitors. There is no difference between these two modes as far as the data that are collected is concerned. For normal responses from the remote units, the MONITOR mode just listens for a response from any remote unit while the POLLER mode sends a POLL message to a selected remote unit and listens for a response from that specific remote unit. One, and ONLY one, master is designated poller on any given segment to avoid message collision on the poll/response party line. The master that is in POLLER mode on a given segment has primary responsibility for that segment.

NILL - A PASCAL constant defined in the TRAMCON software to represent the undefined condition for certain integer subrange variables. For example, an element in the array "linkend-info" is defined if it has a positive value. That same element is NOT defined if it has the value "nill".

POLLER - See the definition of MONITOR above. Along with sending polling messages, the Master that is in POLLER mode on a given segment has the capability to switch remote relays (see definition below) on that segment.

REMOTE UNIT - The TRAMCON system consists of the master units (HP-1000 minicomputers) and REMOTE UNITs. The remote units report alarm/status information to the central master unit which, in turn, displays the information to the TRAMCON operator. Currently, TRAMCON uses two models of a remote unit known as a DATALOK10, the DATALOK10 1D and the DATALOK10 1E. Both models are capable of monitoring up to three sets of communications equipment and one set of site equipment (e.g. batteries, door, etc.). To observe the information from a particular remote unit, the TRAMCON operator specifies the three letter designator for that remote unit. These three letter designators, also called Site Codes, are assigned to each remote unit by the communications system managers and must be unique for each location.

SEGMENT - The communications network that TRAMCON monitors is divided into manageable portions called SEGMENTS. Each segment consists of a collection of remote units connected to the TRAMCON master computer via an RS232, 300-bps party line. To select a particular set of alarm/status indicators, the TRAMCON operator must specify both a segment name and a remote unit name. For operator convenience, the segments are given a short name (up to six characters) in addition to their long names.

SITE - The physical location at which one or more sets of communications equipment are being monitored is called a SITE. Each site can consist of any number of sets of communications equipment that are monitored by TRAMCON.

TRUNK - A logical communication path for 24 channels (T1) is given a TRUNK identifier and starts at a given location (SITE), travels through a specified sequence of sites, and ends at a specified location.

TRANSMISSION MONITOR AND CONTROL SOFTWARE REFERENCE MANUAL

Richard N. Statz[*]

This manual describes the functions of the TRAMCON (TRAnsmission Monitor and CONtrol) On-Line software and the steps necessary to maintain this software. This document emphasizes the software semantics rather than the syntax. The structure of the software is described and the design, and development strategies used in the creation of the software is explained. This manual is intended to provide assistance to experienced programers who want to change or enhance the TRAMCON On-Line software.

Key words: automated technical control; data archiving; Digital European Backbone; microwave radio; network management; polling; pulse count; software; TRAMCON; transmission monitor and control.

1. INTRODUCTION

As any software system matures and passes from the development stage to the maintenance stage, the need arises for a document that summarizes the structure of the software and explains the design and development strategies used to produce the software. This manual is just such a document for the software system known as TRAMCON. Since this document is a summary, by definition, it must be produced after the software system has matured. The larger the software system, the more voluminous is the normal accompanying documentation. This summary document serves as a condensation of the information contained in the other volumes. Since it is produced after the software has matured, any changes from the original design or errors in the original documents can be clarified and/or corrected in this document. The summary document also serves as a quick reference. For answers to most questions concerning the software, this is the document that should be examined first. As the software was developed, information on the use of development tools and procedures for development were sometimes difficult to find or, occasionally, incorrectly documented or not documented at all. This manual carefully documents these hard to find items and procedures.

This manual is intended to supplement rather than replace the Computer Program Configuration Item (CPCI) documents already produced and delivered to the sponsor's software maintenance organization. Certain portions of this document may be a repetition of information contained in those CPCI documents. If so, it is presented in a slightly different fashion or from a different viewpoint. This document describes the philosophy upon which the software was written and the environment necessary to maintain and enhance the software. The programer should consult the CPCI documents for detailed objective discussions of the software. Lastly, it is the intention of the

[*]The author is with the Institute for Telecommunication Sciences, National Telecommunications and Information Administration, U.S. Department of Commerce, Boulder, CO 80303-3328

author that this document serve as a concise description of the Transmission Monitor and Control (TRAMCON) software, placing answers to programers' questions at their fingertips. It is assumed that the user of this manual is familiar with the Hewlett Packard (HP) 1000 minicomputer and the terminology used by HP. Information that can be found in more detail in the HP documents or the CPCI documents is cross referenced in this manual.

2. OVERVIEW OF THE TRANSMISSION MONITOR AND CONTROL (TRAMCON) SYSTEM

The TRAMCON monitoring system was developed to reduce the per-channel-mile cost of communication services by increasing the productivity of the communication operation and maintenance personnel. It was also developed to accommodate the introduction of a new maintenance philosophy required by the upgrade from analog to digital communications equipment. The TRAMCON system is a minicomputer-based data collection system that collects alarm/status and performance parameter data from long-haul telecommunications equipment and presents this information to the operator both on video display and hardcopy devices. In addition, at the operators direction, the TRAMCON system can remotely reconfigure the communications hardware via relays. Figure 1 is a block diagram of the TRAMCON system.

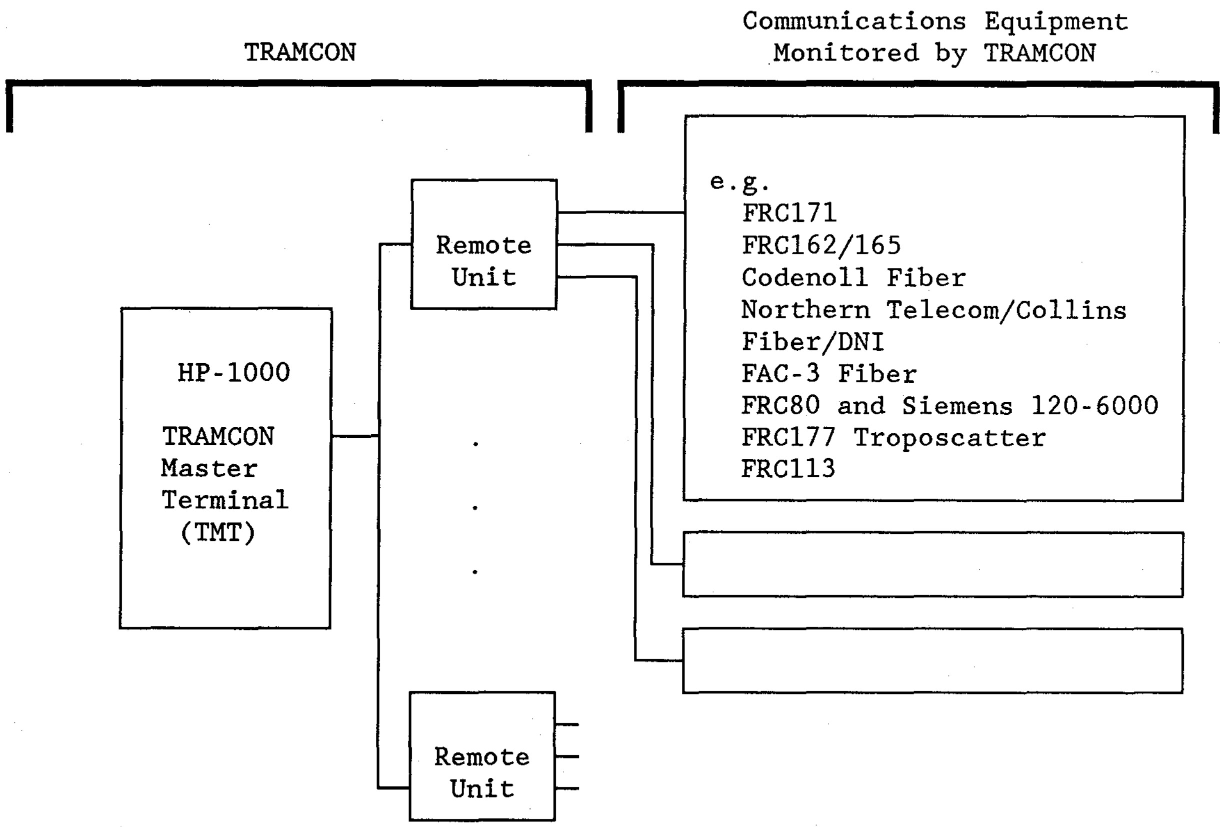

TRAMCON
Communications Equipment Monitored by TRAMCON
HP-1000
TRAMCON Master Terminal (TMT)
Remote Unit
Remote Unit
e.g.
FRC171
FRC162/165
Codenoll Fiber
Northern Telecom/Collins Fiber/DNI
FAC-3 Fiber
FRC80 and Siemens 120-6000
FRC177 Troposcatter
FRC113

Figure 1. TRAMCON system diagram.

Data collection devices, known as "remote units", are connected to the alarm/status points of the communications equipment that TRAMCON is monitoring. Currently, each PHYSICAL Remote Unit is limited to monitoring up to three sets of communications equipment by the software. For each unique set of communications equipment monitored, there is an EQUIPMENT record in the data base. These equipment records give English names to the alarm/status indicators for each kind of equipment.

For every place that a given communications equipment type, such as a DRAMA radio (FRC171), is monitored, there is a pointer to the equipment record that describes that equipment. Since all these pointers are to the same equipment record, the software produces the same results for all DRAMA radio installations. To allow for some uniqueness for any given installation, the "**specific_name**" feature was built into the software and data base. Using Specific Names, the data base Configurator can tailor the general communications description to any specific location. For example, a DIGROUP alarm on port 3 of the multiplexor can have a unique name for any given location depending on the DIGROUP that happens to be using port 3. At some locations, port 3 may even be used currently. Therefore, there would not be a DIGROUP alarm for port 3. In the first case, the DIGROUP alarm for port 3 can be reported with the identifier of the DIGROUP that is currently using port 3. In the second case, the software can be told to ignore the port 3 DIGROUP alarm at the given location by setting a nill pointer in the Specific Name field.

A similar feature to Specific Names has been added to the remote relay function to allow the data base configurator to define relay functions at some locations and not at others.

To allow the TRAMCON software to monitor a new type of equipment, the data base manager simply creates an equipment record that describes the new equipment. Figure 1 lists the communications equipment currently monitored by TRAMCON.

All of the alarm/status information that is collected by these remote units is reported to the HP-1000 minicomputer which, in turn, analyzes the data according to its data base description and produces displays for the TRAMCON operator. Each minicomputer is capable of monitoring up to two sets of remote units consisting of up to 21 remote units each. Combining these numbers with the limit of three sets of communications equipment per Remote Unit, we see that it is possible that a single TRAMCON operator may be responsible for up to 126 sets of communications equipment and up to 42 sets of site equipment.

Currently, operations and maintenance of the analog communications system are performed on a site by site basis. That is, operations personnel are stationed at every communications site and are responsible for the equipment at their own site. This is costly, especially when the site is a simple repeater located on a difficult-to-access mountaintop. The TRAMCON system allows an operator to maintain not only the equipment on site, but all of the equipment at the local site as well as several remote locations. Therefore, the first reduction in manpower cost would result from the unmanning of the most inaccessible sites. From an active and well-staffed central location, the unmanned mountaintop sites can be watched, their problems can be diagnosed, switches can be made from malfunctioning equipment to redundant functioning equipment, and the proper maintenance crews can be dispatched if necessary.

The intent of TRAMCON is not to displace or reduce the responsibility of human operators but rather to increase their span of control by placing information of a more refined quality at their disposal to allow them to make better control and troubleshooting decisions.

The functional requirements of the TRAMCON system were developed over a number of years, starting with a very basic list of features. As the operators gained experience with the initial installation (known as the Enhanced Fault Alarm System, or EFAS), enhancements of display capability and of monitoring function were added at their suggestion. In parallel with the installation of the digital communications system known as the Digital European Backbone (DEB) and as a result of changes in system control philosophy, the official functional requirements for TRAMCON were articulated by the Defense Communications Agency. This began in 1981 and culminated with the issuance of a Concept of Operation in August 1984.

The basic functional requirements for the TRAMCON system, consisting of master and remote units, are as follows: remote unit polling, alarm scanning at the remote location, isolation of alarms to a particular location and equipment, control of specified functions at a remote location, monitoring of specified performance parameters at remote locations, presentation of the information received from remote locations to an operator and to other elements of system control as required, and allowing at least two master units to share responsibility for any particular segment of the network.

To perform the functions described, a minicomputer handles the master unit functions and does the required data processing to put the information into the desired formats for presentation to the operators. This minicomputer, referred to as the TRAMCON Master computer, sends poll messages to remote units and receives and decodes the responses. The information in the responses is processed to generate displays on a CRT terminal. These displays present alarm, equipment status, and performance parameter data gathered from the remote sites in easily understood English text formats designed to show individual site and entire system conditions.

To improve the reliability of the monitoring system, at least two master units are able to monitor each segment or group of communication sites. At any given time, only one of the masters monitoring a particular segment will be designated as the POLLING master for that segment. The polling master will actually control the remote units on a segment while any other master monitoring that segment will operate in a listen-only mode. The polling line is a party line and is broken at segment boundaries so that only polling messages intended for a particular segment's remote units appear on that segments poll line.

The TRAMCON On-Line software system was based on an existing system called the Enhanced Fault Alarm System (EFAS). The general concepts listed below were continued from EFAS to TRAMCON.

1. Remote units (data collection devices) reporting to a central master computer over an RS-232 party line.
2. The remote unit network is divided into segments that are sets of remote units.
3. Each segment has at least two masters assigned to monitor it with one master sending the polling messages and both listening to the responses.
4. The information collected by a remote unit is categorized as
   - A. Two-state - single bit ON or OFF
     LATCHING alarm or MOMENTARY (nonlatching) status
   - B. Analog - Voltages encoded into 8-bit bytes
   - C. Digital - Pulses counted and encoded into 8-bit bytes

   The analog and digital data are calibrated and converted into units familiar to the operator. The operator is allowed to adjust this calibration and to set operating thresholds for these values.
5. Radio or communication equipment being monitored can be reconfigured remotely by the operator by activating relays in the remote unit that are, in turn, wired to switches and relays in the radio equipment.
6. The information collected by remote units is formatted and presented to the operator on a terminal display device and/or hard copy device. This information is requested by the operator by command entry via a standard ASCII keyboard.
7. Several operator consoles can be connected to a given master.
8. Alarm/status data is archived on the disc and this history is made available to the operator.
9. The master system has a battery-backed-up time/date clock so that the system can survive power fluctuations and outages without having to be restarted.

As indicated by the list above, the TRAMCON system had a solid, well-developed foundation, and the functionality of TRAMCON does not differ greatly from that of EFAS. A few major design improvements were made though, based on experience with the EFAS system.

The primary improvement is in the TRAMCON software's ability to monitor virtually any kind of communication equipment without a change to the software. As a matter of fact, the TRAMCON software can monitor ANY kind of equipment that is able to supply the types of data listed in item 4 above. As an example, the TRAMCON system monitors many non-communication indicators such as the door sensor or the generator fuel gauge. Although the EFAS system possessed this ability to monitor any equipment, a software change was required to introduce new equipment into its repertoire. Everything about any of these alarm/status values, including the English names, are hardcoded in the software.

To allow TRAMCON to monitor any equipment that is introduced in the future without making a change to the software, the TRAMCON system was designed with two main parts. The On-Line software performs the actual data collection and display functions of TRAMCON and relies on a configuration data base rather than hard code to define its environment for any particular master. The Configuration software (Configurator) maintains this data base for all masters. Information that could change at any time, such as segmentation, site names, alarm names, or information that differs from master to master is placed into this Configuration data base. Any aspects of the system that are the same on all masters, such as display formats or operator command mnemonics, are incorporated into the TRAMCON On-Line code.

The TRAMCON hardware/software system development would not be complete without a comprehensive document that describes the details of the software structure and the software development procedures necessary to maintain and enhance the TRAMCON system. With every software system there is guaranteed to be unique, poorly documented, and/or non-documented development procedures that need to be brought to the attention of software maintenance people. Also, during the initial software development, problems were encountered that required a work-around or a correction from the vendor. These problems, along with their solutions, must be documented to prevent further waste of valuable development time rediscovering the solutions.

This manual is presented by the TRAMCON developers as an example of such a comprehensive, timesaving document. Further maintenance and enhancement of the software system which this document describes would not be practical without the information that places at the programmers fingertips.

The structure and organization of the manual is also critical to its utility. Some of the information is a repeat of that contained in the volumes of documents supplied by the software developers and the computer vendor. The information is repeated here to make it more accessible to software users and to discuss certain software tools and procedures as they pertain to a specific application.

The author believes that the reader will discover, in this document, a valuable guideline for the production of a similar document for any software system they may develop and that no software system is complete without such a manual. Further, this document cannot be properly completed until the software system is fully developed and the initial version is ready to be fielded.

3. THE TRAMCON ON-LINE SOFTWARE SYSTEM ENVIRONMENT

This section describes the minimum hardware and support software (operating system) required by the TRAMCON on-line software. Hardware resource allocation strategies that might optimize the efficiency of the TRAMCON on-line software are discussed.

3.1 Hardware Configuration

The hardware constituting a TRAMCON data collection system is shown in Figure 2. Most hardware was purchased from one vendor to reduce the maintenance and logistic support problems that naturally result from multiple vendors. The primary vendor was Hewlett Packard. Any hardware item that is not HP was purchased from someone else only because HP did not make that item, or that item was already being used in the predecessor system to TRAMCON known as EFAS. The only items currently not HP are the modems and the data collection device built by Pulsecom Corporation known as a DATALOK10. Figure 2 shows the most recent TRAMCON hardware configuration.

Briefly, the hardware chosen for the development and implementation of the TRAMCON master unit was the HP-1000 F-Series minicomputer running the RTE-6/VM operating system version A.85. The operating system version name was derived from its release date of first quarter (A) of 1985. The original system disc drive was the HP7906 20-Mbyte drive which has been replaced by the 65-Mbyte Winchester technology HP7912 drive with built-in streaming cassette tape drive. The new drive has already been incorporated into the TRAMCON system and will, therefore, be the hardware discussed in this manual.

With the newer releases of the RTE operating system, the user was offered two choices of disc file systems. The older and more established file system is known as File Manager (FMGR) and the new system, which introduces the more advanced hierarchical structured file system, is known as Command Interpreter (CI). After careful consideration and consultation with HP, the choice was made to use FMGR alone for the development of the TRAMCON system. Because of its advances in flexibility and structure, CI would have been the clear choice. But, the RTE operating system running on the F-Series machine could not be completely divorced from the FMGR system. For the convenience and simplicity of working with and maintaining just one file system, the choice was made to stick with the required FMGR system. Therefore, all references to file names and disc file manipulation throughout the rest of this document are with regard to the FMGR system and require reader familiarity with that system only.

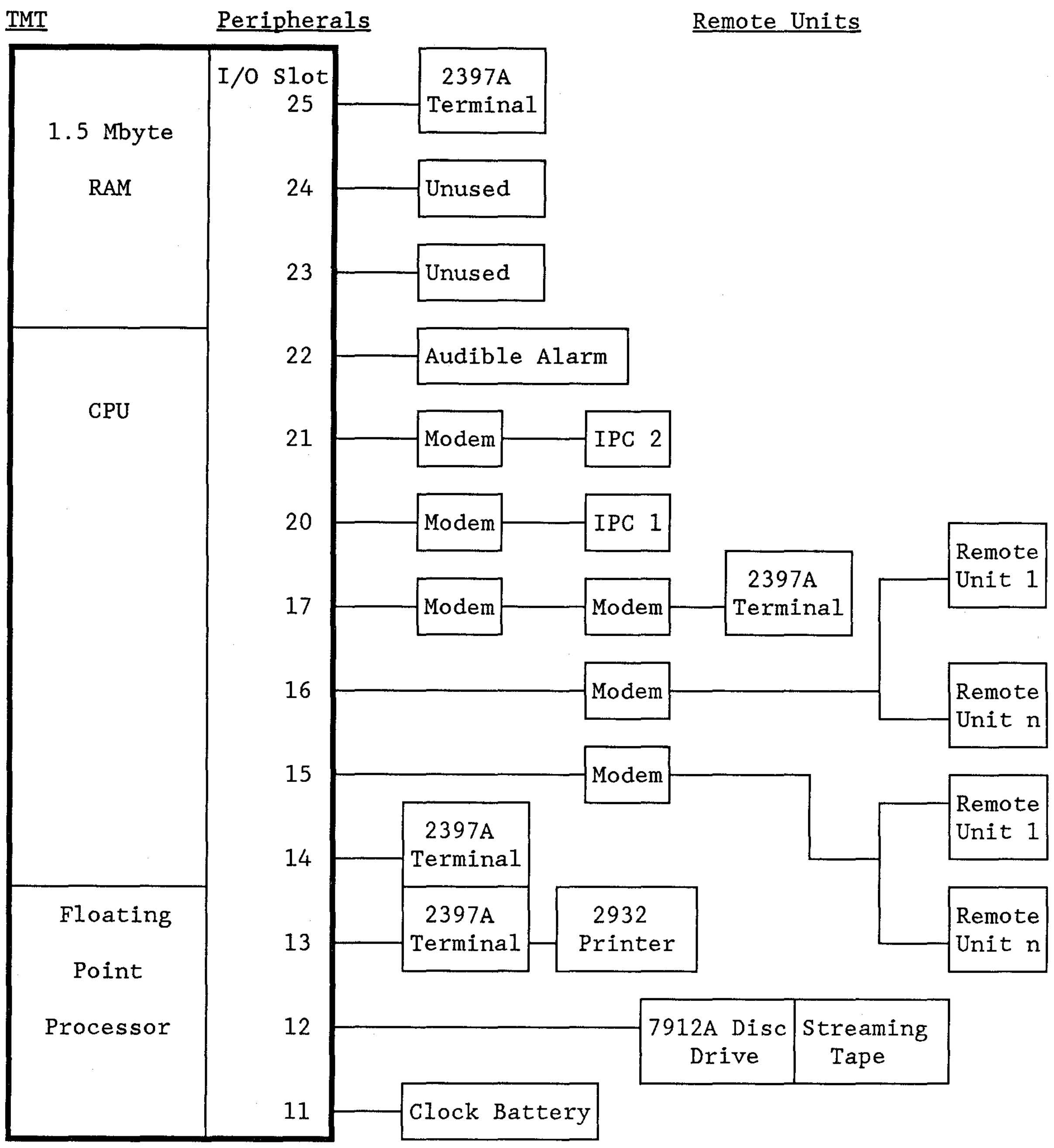


Figure 2. TRAMCON Hardware diagram.

Figure 3 expands the TRAMCON master computer backplane description and lists the logical unit (LU) numbers and equipment numbers assigned to each input/output (I/O) slot (select code) in the backplane of the computer. With the TRAMCON master computer, there is a one-to-one correspondence between I/O slots and equipment numbers. This correspondence is set up at system generation time (refer to Section 10) and cannot be altered without regenerating because the equipment table is a fixed part of the operating system. What can be changed, at any time, is the association between the equipment and the LUs. This flexibility allows software to refer to different equipment without being rewritten. The software does its I/O by referring to LU numbers that, in turn, can be associated with a different piece of equipment each time a program is executed.

Select Code	Interface	Driver	Logical Unit(s)	Equipment Number
4	Power Failure	DVP43	13	26
10	Floating Point Processor	N/A	N/A	N/A
11	Hardware Time/Date Clock	DVT43	31	18
12	7912A Disc Controller	DVM33	2,10	1
13	Terminal (System Console)	DVX05	1,25	2
14	Terminal (Segment Console)	DVX05	26	3
15	Segment 0 Poll Line	DVA76	15	4
16	Segment 1 Poll Line	DVA77	16	5
17	Terminal (RDT)	DVX05	27	6
20	IPC Channel 1 (HDLC)	DVA66	17	7,8
21	IPC Channel 2 (HDLC)	DVA66	19	9,10
22	Digital Output (Audible)	DVS72	14	11
23	Unused	-----	--	--
24	Unused	-----	--	--
25	Terminal	DVX05	28	13

Figure 3. TRAMCON master computer backplane layout.

Also permanently associated with the I/O slot is the software driver that processes the I/O requests for each kind of equipment. Only two of these drivers, DVA76 and DVA77, are not standard device drivers. These two drivers were written by the TRAMCON developers and are described in detail in Section 6.3. Any of the drivers can be updated and tested on-line by following the procedures detailed in Section 8.2.6.

Because the IRU was planned from the start, TRAMCON software was designed to support virtually any remote unit device with a minimum of additional code necessary for this support. Code is already in place to support the now-defunct IRU. This ability to support ANY remote unit feature is currently demonstrated by the support of two models of the DATALOK10: model 1D and model 1E. Because the DATALOK10 is dumb, each model generates a fixed response that is organized differently. Separate code is needed to process the different responses and convert them into a generic response that the TRAMCON On-Line software can analyze (refer to Section 6).

As shown in Figure 2, the other major component of the TRAMCON hardware is the TRAMCON Master Terminal (TMT). The TMT consists of a minicomputer, several terminal display devices, a disc memory system with integrated streaming tape, and several hardware interfaces that allow the TMT to communicate with all its peripheral devices, with the network of remote units, and with each other. The central processor portion of the TMT is an HP-1000 model 2117F, also known as an F-Series computer. One Mbyte of central memory has been installed and another .5 Mbyte is planned.

The basic hardware has been upgraded only when absolutely necessary to avoid costly documentation changes and to avoid having more than one fielded configuration. In other words, uniformity of both hardware and software was a primary consideration when designing and implementing the TRAMCON system. The first item upgraded was the HP-2647F display terminal. The primary reasons for replacing the HP-2647F were cost and obsolescence. Hewlett Packard no longer manufactures this terminal model and has replaced it with better, less expensive models. Since there were so few HP-2647F terminals fielded when the decision to change was made, for uniformity all HP-2647F terminals were replaced with newer models.

The next major item replaced was the HP-7906 disc drive. The primary reasons for this decision were obsolescence, high failure rate in the field, and a need for larger capacity. The HP-7906 disc was replaced by the HP-7912 disc, increasing the capacity from 20 Mbytes to 65 Mbytes and, hopefully, decreasing the failure rate by using a Winchester-technology device that is sealed from the outside environment. Again, for uniformity, the 13 HP-7906 units already fielded were retrofitted with the new disc.

The last item mentioned here is an addition rather than a replacement. The increase in memory capacity to 1.5 Mbytes allows software designers some flexibility in allocating memory to reduce possible bottlenecks caused by program swapping and data storage and retrieval on disc. The 1-Mbyte capacity was technically not enough for a fully configured TRAMCON master computer with two segments or each, with 21 remote units.

3.2 Memory Allocation

As mentioned in the preceding section, the central memory will be increased from 1 Mbyte to 1.5 Mbytes. The reason for the increased memory is that the current system uses all of the 1 memory currently available. This allows absolutely no room for future enhancements. Even in the 1.5 Mbyte system allocation proposed above, compromises in system performance were made because of a shortage of memory. Examples of compromises follow: fewer time-critical programs are made memory-resident, not enough large segments exist to accommodate heavy (multiple terminal) activity, and real time data are not being kept in memory.

The memory has been repeatedly repartitioned to achieve the best performance possible, given the already constraining size. The software may technically function after an inordinate amount of massaging of the memory allocation, but it does not function nearly as well as it can on the HP-1000.

Always, the overriding bottleneck is disc I/O. To reduce I/O to a minimum, among other things, program swapping must be kept to a minimum. That means making the time-critical programs, such as the polling processors, lock themselves into a section of memory that no other software module can use. Also, data that are often referenced should be memory-resident. As a general rule, the more memory, the greater the reduction of costly disc I/O.

The following discussion details the information upon which the decision for more memory was based. The memory can be divided into three major components: (1) the operating system, (2) the TRAMCON application programs, and (3) shared data.

First, we will look at the shared-memory requirements for the data base. The following is a list of major data base components and their sizes.

	Record name	Bytes	16-bit words
	heap	8,670	4,335
	master	94	47
	network	280	140
	links	3,500	1,750
	dictionary	14,000	7,000
*	crt	12	6
	segment	288	144
	trunk	152	76
	equipment	4,626	2,313
	site	18	9
*	remote	30	15
*	link_end	238	119
*	remote_status	2420	1210
*	link_status	1,528	764

* Dynamically allocated only when non-NIL pointer encountered.

The records heap, master, network, links, and dictionary are the same size for all TRAMCON masters. All other items listed above must be multiplied by some factor depending upon the data base configuration for the particular master. The following shows the storage requirements for a fully-configured TRAMCON master and the factors used to derive these requirements:

Factors for a fully configured system

2	Segments per master
256	Sites per master
21	Remote Units per segment
4	Categories per remote (3 link plus 1 site)
30	Parameters per category
100	Trunks per segment
20	Equipment records per master
5	Terminals per master

Item size x factors listed above

	Bytes	16-bit words
Fixed data	26,544	13,272
Terminal x 5	60	30
Segment x 2	576	288
Trunk x 100 x 2	30,400	15,200
Equipment x 20	92,520	46,260
Site x 256	4,608	2,304
Remote x 21 x 2	1,260	630
Link_end x 4 x 21 x 2	39,984	19,992
Remote_status x 21 x 2	101,640	50,820
Link_status x 4 x 21 x 2	256,704	128,352
Total Requirements	554,296	277,148
Total Memory Available	512,000	256,000

The factors listed above are constraints of the configuration data base as defined in the software in module [RECR3 (refer to Section 11.1). If any of these values are changed, the TYPE or CONST definitions in [RECR3 would have to be modified and ALL TRAMCON software modules would have to be recompiled and reloaded to incorporate the change.

The item sizes are multiplied by the appropriate factors to give the actual memory requirements for a fully-configured system. A fully-configured system has a very low probability of occurring in the real world. Therefore, the shared memory requirements were based on the much more realistic average system specified below. Some of the factors will be smaller for the average system. For example, the average system will have only 3 categories per remote unit rather than the maximum 4 categories.

Factors for the average system

2	Segments per master
10	Remote Units per segment
256	Sites per master
3	Categories per remote unit(2 link plus 1 site)
30	Parameters per category
40	Trunks per segment
4	Equipment records per master
2	Terminals per master

Item size times factors listed above

	Bytes	16-bit words
Fixed data	26,544	13,272
Terminals x 2	24	12
Segment x 2	576	288
Trunk x 40 x 2	12,160	6,080
Equipment x 4	18,504	9,252
Site x 256	4,608	2,304
Remote x 10 x 2	600	300
Link_end x 3 x 10 x 2	14,280	7,140
Remote_status x 10 x 2	48,400	24,200
Link_status x 3 x 10 x 2	91,680	45,840
Total Requirements	217,376	108,688
Total Memory Available	500,000	256,000
RTE-6 Operating System	162,000	81,000
TRAMCON Program Space	350,000	175,000
Total	512,000	256,000

Based on the memory requirements for the maximum and average systems which were just discussed, the following shows the currently employed memory allocation scheme for the 1 Mbyte system and a proposed scheme for the 1.5 Mbyte system:

	Current	Future
Operating System (RTE6/VM, Ver A.85)	204K	204K
Shared memory for configuration and dynamic data storage		
Miscellaneous (e.g., dictionary, crt info)	86K	100K
Dynamic alarm/status (current space already inadequate for 2 segments of 21 remotes each requiring 300K minimum)	294K	350K
Time-critical programs		
DS programs		30K
PLRP	94K	100K
MTRP	94K	100K
CMMD		64K
UP		52K
POLL		50K
Fault isolation programs		64K
Partition for large segmented programs		130K
Partitions for display programs (one per CRT)	228K	256K
Total	1 Mbyte	1.5 Mbyte

In the memory allocation scheme laid out above, the additional .5 Mbyte of memory primarily would be used to place more of the critical software into memory (i.e. make memory-resident). This would reduce disc swapping of program modules that run periodically and thus increase the efficiency of the TRAMCON On-Line system.

3.3 Program Residency

The most common bottleneck in most computer systems is the I/O between the CPU and the mass storage device--in TRAMCON, the 7912 disc. In a multi-tasking system with limited central memory, the disc I/O involved in swapping programs can be significant and becomes a prime target for performance improvement adjustments. The tradeoff here involves memory space versus swapping delay. The ideal would be to have all the On-Line software memory-resident, which would require NO program swapping. This extreme is not feasible as illustrated in Section 3.2. The following table shows the memory partitioning scheme currently used for the TRAMCON field system:

Partition number	Size (1024-word pages)	Use
1	32	Disc I/O Manager (D.RTR)
2	5	Small, periodic programs
3	5	Small, periodic programs
4	12	Small-medium programs
5	14	Small-medium programs
6	27	Small-medium programs
7	32	Full-size programs
8	32	Full-size programs
9	46	Program MTRP
10	46	Program PLRP
11	190	Shared data (EMA)

The memory allocation (partitioning) is specified to the system generator (see Section 10) and can be adjusted without regeneration by setting bit 5 ON in the S-register when booting the system (refer to the RTE-6/VM System Manager's Reference Manual, Chapter 10).

The order in which partitions are defined is significant. The order shown above ensures that the proper programs are assigned to proper partitions by the scheduler. The program scheduler starts with partition 1 and searches for the first partition large enough to hold the program being scheduled. Defining the smallest partitions first ensures that the program which best matches the partition's size is assigned to it.

Memory must first be allocated for the shared data. The more data that is stored in memory, the less disc I/O that is involved in accessing the data. Any memory left over can be allocated as program space for execution of the TRAMCON On-Line programs.

A few small partitions were defined to hold small Distributed Systems (DS) programs, such as UPLIN, and small TRAMCON On-Line programs, such as UP, that run often and on a regular basis. The two main remote unit response processing programs, MTRP and PLRP, were made memory resident to avoid segment swapping for these extremely time-critical programs. Two partitions (9 and 10) were set aside exclusively for these two programs. Both MTRP and PLRP were segmented with all segments declared as memory-resident segments. This step ensures that the entire program will be placed into memory any time it is executing so that segment overlaying does NOT involve disc I/O, merely memory map swapping. However, this step does not prevent the entire program from being swapped and having to share its memory partition. Therefore, when these programs are scheduled and are assigned the proper partitions, they immediately lock themselves into memory (refer to RTE-6/VM Programer's Reference Manual, p. 2-70, Program Swapping Control - EXEC 22) to prevent the operating system from doing a complete program swap with them.

4. PROGRAM SCHEDULING

From a scheduling standpoint, each TRAMCON program falls into one of two general categories. Some programs, such as the remote unit response processing programs, are scheduled once at bootup and never terminate. Others, such as display producing programs, are scheduled on operator demand, run to completion, and are rescheduled at some later date. This section discusses the scheduling of the various TRAMCON programs.

4.1 TRAMCON System Initialization - INIT

The HP-1000 minicomputer is restarted from a halted state by following the bootup procedure specified in the TRAMCON (version 1.8) Operator's Manual, Section 6.4. The HP-1000 minicomputer can be booted from several different devices, but the field TRAMCON system is booted from the 7912 disc. The bootup device is specified by the contents of bits 14 and 15 of the S register as specified in the procedure. The bootup routines for each of the possible devices are stored on ROMs located under CPU cards just behind the front panel. The bootup ROM for the TRAMCON field system is located at address 3, so both bits (14 and 15) are set ON. This bootup is referred to as a SOFT boot because it is assumed that the RTE operating system software and the TRAMCON software are intact on the 7912 disc and the disc system is operational. The TRAMCON field system bootup process is diagrammed in Figure 4.

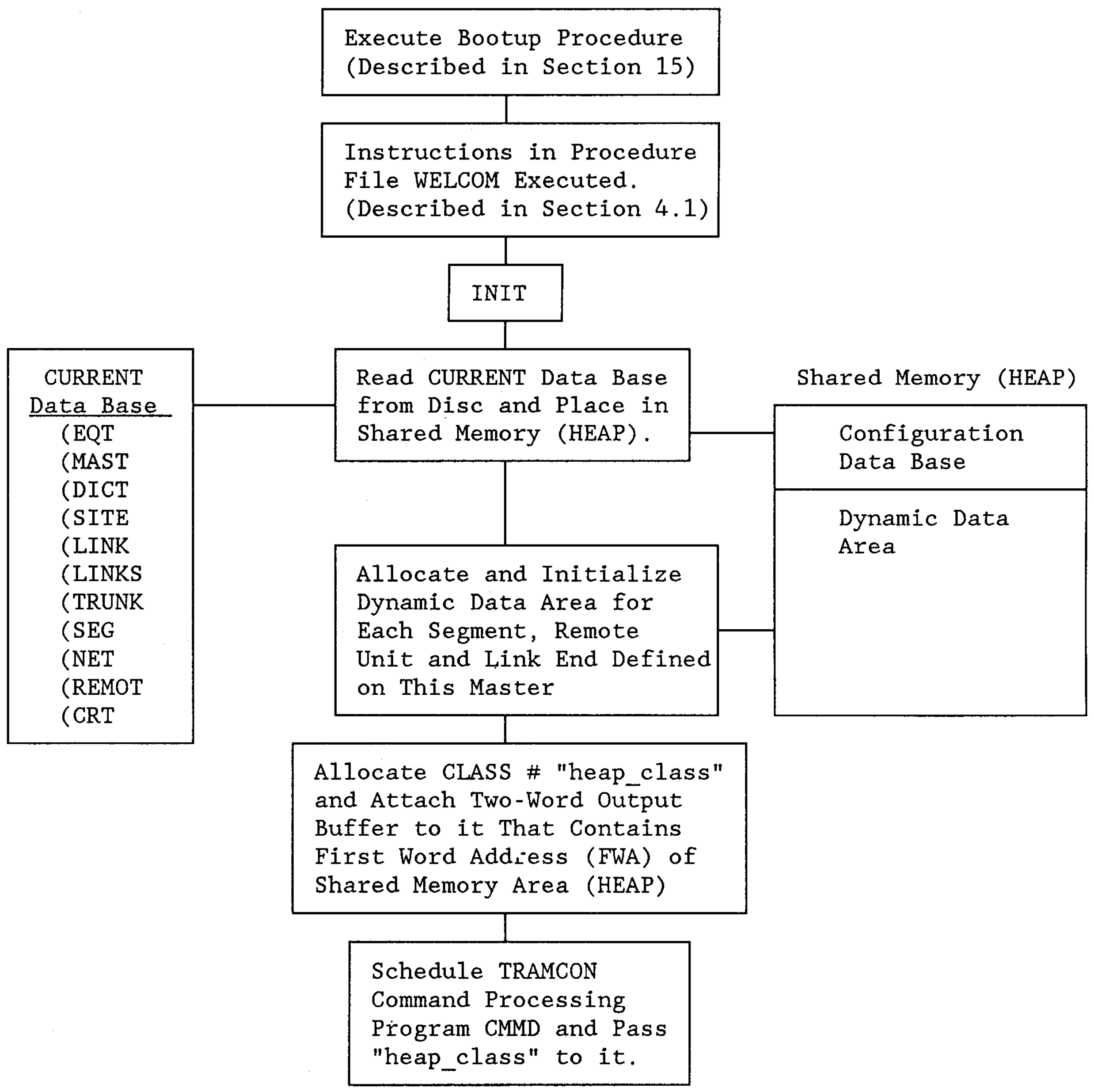

Execute Bootup Procedure (Described in Section 15)
Instructions in Procedure File WELCOM Executed. (Described in Section 4.1)
INIT
CURRENT Data Base
(EQT
(MAST
(DICT
(SITE
(LINK
(LINKS
(TRUNK
(SEG
(NET
(REMOT
(CRT
Read CURRENT Data Base from Disc and Place in Shared Memory (HEAP).
Shared Memory (HEAP)
Configuration Data Base
Dynamic Data Area
Allocate and Initialize Dynamic Data Area for Each Segment, Remote Unit and Link End Defined on This Master
Allocate CLASS # "heap_class" and Attach Two-Word Output Buffer to it That Contains First Word Address (FWA) of Shared Memory Area (HEAP)
Schedule TRAMCON Command Processing Program CMMD and Pass "heap_class" to it.

Figure 4. TRAMCON On-Line field system initialization.

The bootup ROM program transfers to a larger bootup program found at a fixed location on the bootup device (the 7912 disc for the TRAMCON system) to the HP-1000 central memory starting in memory location 2, then branches to the program just loaded. This larger bootup program proceeds to load the memory-resident portion of the RTE-6/VM operating system into memory. Once the loading of the operating system is complete, control is transferred to the RTE operating system program scheduler, which schedules the program FMGR. Disc files called PROCEDURE files can be created to contain commands for the FMGR program. When the FMGR program is initially scheduled, it is told to look in disc file WELCOM, which is one such procedure file, for its initial instructions. This WELCOM file facility allows the user to perform virtually any function, automatically, when the system is booted up. The WELCOM file used for the TRAMCON field system is listed in Figure 5.

```
:SV,4,,IH
:SYCU,ON
:CT,1,30B,417B
:CT,1,31B,1
:RU,PAKLU,QT
:IF,1P,EQ,0,2
:PK,1P
:IF,,EQ,,-4
:RU,SETCL
:RU,SETCR
:RU,CLNUP,ALL
:RP,EDITR
:RP,PROGL
:RP,VCPMN
:RP,LUQUE
:RP,SYSAT
:RP,RSM
:RP,#SEND
:RP,RFAM
:RP,DLIST
:RP,DSINF
:RP,OPERM
:RP,EXECW
:RP,EXECM
:RP,PTOPM
:RP,REMAT
:RU,DINIT,(DINIT
:SYAG,100
:TR,RUINIT
```

Figure 5. TRAMCON initial bootup procedure file WELCOM.

The first FMGR command, "**SV,4,,IH**", tells FMGR not to display any messages that do not concern fatal problems. The next command, "**SYCU,ON**", causes the execution of the diagnostic program CU, which displays CPU usage as the contents of the S register on the front panel. The more the CPU is being used, the more lights are set ON in the S register. The next two commands, "**CT,1,30B,417B**" and "**CT,1,31B 1**", are used to configure the RS-232 interface associated with LU 1 (system console) and are used only when the system console is connected to the interface via a modem cable (12966-60006).

The next four statements, "**RU,PAKLU,QT**", "**IF,1P,EQ,0,2**", "**PK,1P**", and "**IF,,EQ,,-4**", will automatically pick all the disc LUs defined in the system. The data storage on the disc becomes fractured (data interlaced with unused portions of the disc) as files are created, deleted, or replaced. In the TRAMCON field system, there should be NO need for this packing if the disc is packed before any new software is shipped. In normal TRAMCON field operation, NO disc files are created or deleted and only the Configuration data files are replaced. If these files were placed at the end of disc LU 10 and purged before the new data base is placed onto LU 10, NO fracturing would occur. The disc space released by purging files at the end of the used portion of any disc LU is automatically recovered.

The next statement, **"RU,SETCL"**, executes the small program SETCL which sets the software time/date clock from the battery-reinforced hardware clock. During normal operation (when the machine is not halted), the time/date is derived from the software clock, which is maintained in memory by 10 ms interrupts on the time base generator interface in Select Code 11_8. If the hardware time/date clock loses the correct time/date, program SETCL can be used to set the hardware clock from the software clock by issuing the statement, **"RU,SETCL,-1"**. This assumes that the software clock is set to the correct time/date. The software clock can be set with the operating system TM command (refer to RTE-6/VM Quick Reference Guide, p. A-8).

The next statement, **"RU,SETCR"**, executes the small program SETCR to set the cartridge ID for disc cartridge 10 to SYSTEM so that the DS software can access files on this cartridge. The cartridge ID is mentioned in the RTE-6/VM Programer's Reference Manual, p. G-4. On any TRAMCON system, this flag must be set to SYSTEM, but occasionally during software development it could inadvertently be altered. Nothing is lost by running this program.

The next statement, **"RU,CLNUP,ALL"**, executes the small program CLNUP to search disc directories for files that might have been left open when the system halted. Any file found open to a program that is no longer running, will be closed. At this point, the WELCOM file is open to FMGR which, of course, is running and therefore, this file is NOT closed. This program was acquired from the HP-1000 user's group.

The next 15 **"RP"** statements make the 15 DS programs available to the operating system by loading their executable codes into the system scratch area and assigning a program ID segment to each of them.

Following the "RP" statements, **"RU,DINIT,(DINIT"**, executes the program DINIT, to initialize the DS software and InterProcessor Communication (IPC) network. The program DINIT is an interactive set of questions about the network configuration and answers to these questions are found in disc file (DINIT. Most of the answers in (DINIT are generic, or the same, for all TRAMCON masters. Only a few of the answers, such as **"Local Node Number"**, are unique for each master. All of the answers for each master are known to the Configurator program. File (DINIT is automatically generated by the Configurator when it generates a master specific data base (refer to Section 14). The DINIT program establishes a given TRAMCON master as an active node on the IPC network.

Figure 6 shows the information that should typically appear on a system console screen after the programs CLNUP and DINIT have been run. This information will appear briefly before the program INIT clears the screen and presents the TRAMCON logo.

The next statement, "**SYAG,100**", is an operating system command to set the aging delay for program swapping. The value 100 is suggested by HP. Program swapping on the HP-1000 is very elementary, and the setting of the swap delay probably is insignificant.

```
SET TIME
Checking      2
WELCOM::2      is open   shared    to   FMGR    - OK
Checking     10
END DINIT
 DS MSG: LU # 17  JUST CAME UP
 TIME: DAY  214 8 : 19: 23
 DS MSG: LU # 19  JUST CAME UP
 TIME: DAY  214 8 : 19: 23
RFAM: LIMITED DISC SPACE, THE NUMBER OF FILES HAS BEEN LIMITED TO   5
```

Figure 6. Typical bootup messages on system console.

The next statement, "**TR,RUINIT**", is the most important command in the WELCOM file. This statement activates the TRAMCON On-Line software. Technically, this command transfers FMGR command processing control to procedure file RUINIT. This indirect step was necessary during development because program INIT had several options that could be passed to it as run string parameters. All but one of these run string options has disappeared during development. The only remaining option is the third parameter that, if set to 73, will start the TRAMCON On-Line software with the master password entered ("restricted_access" is false, refer to Section 12.1). This is still a desirable convenience for software maintenance, but undesirable in the field.

Program INIT allocates the EMA partition called SHAR1 as the HEAP area that is to be shared by all TRAMCON On-Line programs. The information stored in the HEAP is described in detail in Section 11.1 of this manual, which discusses the INCLUDE module [RECR3. Basically, INIT reads the Configuration data from the 12 Configuration data base files (see Section 11.4) and places the records from each file into the HEAP. The data in these files are hierarchically structured with records from one file containing pointers to records in other files. The Configurator does not know where, in EMA, each of these records will be at the time the TRAMCON software is booted up. All the Configurator program knows is how the records are connected together. For example, if a record such as the master record contains a pointer to a particular site record, the value placed in the field in the master record by the Configurator is the record number corresponding to the desired SITE record in file (SITE. Since these values will be translated by INIT into actual two-word EMA addresses, they are defined as two-word integer values. To accomplish this translation from integer values to EMA addresses, two almost identical TYPE definition modules, [RECR2 and [RECR3, are maintained. The module [RECR2 contains near-duplicates of all the record TYPE descriptions for the Configuration data base records. The only difference between the two definitions is in the TYPES of these record pointer fields. Module [RECR2 defines these pointers to be of TYPE integer and [RECR3 defines these pointers to be Pascal pointer types. As INIT allocates space for these records in the HEAP, it creates a pointer (two-word

EMA address) for each record. Program INIT substitutes the newly created actual memory addresses into the records as it places them in memory.

NOTE

Although the only reference to the INCLUDE module [RECR2 is in the program INIT, it is mandatory that the definitions in [RECR2 and [RECR3 match. Any time that TYPE definitions for any of the 11 Configuration data base disc file record definitions change, the change must be made to both [RECR2 and [RECR3.

After transferring all the Configuration data to the HEAP, program INIT allocates a large portion of the HEAP for the dynamic run-time data described in detail in Section 11.1. Program INIT terminates the FMGR program and schedules the TRAMCON command processing program CMMD, passing to it the CLASS number that leads to the first word address (FWA) of the newly created HEAP.

NOTE

Since program FMGR is terminated by program INIT, any further commands in file WELCOM beyond "**TR,RUINIT**" will NOT be executed.

Program INIT's function is complete and it terminates.

4.2 Programmatic Scheduling

Programs on an RTE system can be scheduled in either of two ways. First, a program can be scheduled directly by using the RTE commands RU or ON. The second scheduling method is programmatic. That is, one program can schedule another by issuing a call to the system routine EXEC. A program may be scheduled by either of the above methods to execute immediately or at some later time. That same program may also be scheduled to execute repeatedly at future fixed time intervals.

The TRAMCON On-Line programs use all of the scheduling methods described above. In normal operation, the operator does not have to explicitly execute any TRAMCON program.

The only program executed by using RTE commands is the TRAMCON initialization program INIT. Even the command to schedule INIT is not entered by the operator. Instead, the program INIT is automatically scheduled by an RU command, which is the only instruction in procedure file RUINIT located on LU 10. The instruction to execute the INIT program is not located directly in the bootup procedure file WELCOM because of development options that can be included in the execute instruction. Currently, the only option remaining is to execute INIT with the "**access_restricted**" password entered. This is done by setting the third run-time parameter to 73 as in "**RU,INIT,,,73**". If this password entry convenience is no longer desired, then the code to process the run time parameters could be removed from INIT, plus the procedure file RUINIT could be removed from LU 10, and the statement "**:RU,INIT**" could replace the statement "**:TR,RUINIT**" in file WELCOM. As mentioned above, the program INIT is the only TRAMCON On-Line program scheduled, explicitly using an operating system command. Program INIT, in turn, schedules the program CMMD as its last function before terminating permanently.

All other TRAMCON programs are scheduled programmatically, and all of them are initially scheduled by the program CMMD. The program HR is initially scheduled by CMMD, runs to completion, and reschedules itself to run on the hour every hour until TRAMCON is halted. Another set of programs is scheduled by CMMD when the system is being initialized. These programs, which include PLRP, MTRP, KYBRD, LOF, LON, and POLL, never run to completion. Rather, they continuously loop, spending most of their time waiting for input from the operator, another device, or another program. The remaining TRAMCON programs are scheduled on random demand from the operator or from another program. Programmatic scheduling is accomplished by calling the system routine EXEC with the function code 24 (see RTE-6/VM Programer's Reference Manual, p. 2-57). This will cause the operating system to place the program to be scheduled into the "**run immediately**" queue. The program scheduling process is summarized in Figure 7.

Before discussing program scheduling further, a few words should be said about program type, which is an important aspect of a program running under the control of the RTE operating system. Program types are summarized in the RTE Programer's Reference Manual, p. D-2. The TRAMCON On-Line programs are type 6. A type 6 program can be kept on disc in executable form without having to constantly occupy a program ID segment. Program ID segments are a finite resource, and are used by RTE to track the status of programs that are currently executing.

There are many more programs in the TRAMCON software system than there are ID segments. For example, although there are 40 ID segments generated into the TRAMCON system, once the continuously-running programs are started, only 10 to 12 ID segments remain for any other programs that may want to run. These programs are randomly scheduled programs such as various display and data transfer programs. How many of these programs will run, and when, depends on operator activity. Program CMMD schedules these programs by calling the routine "**clone_and_run**", which is defined in the system library called TRLIB. Routine "**clone_and_run**" proceeds to actually schedule the given program by performing certain steps. A program is assigned a free ID segment by the operating system through calls to the FMGR routines **OPEN, IDRPL and CLOSE.** Routine OPEN is called to open the type 6 program disc file. Routine IDRPL programmatically performs the FMGR function RP (Restore Program), which places the executable program code into the RTE scratch area on LU 2. Function RP then places the ID segment template, found with the executable code in the opened disc file, into a free ID segment. Finally, the routine CLOSE is called to close the disc file. Now the program is known to the operating system and may be run using the RTE routine EXEC with function code 24.

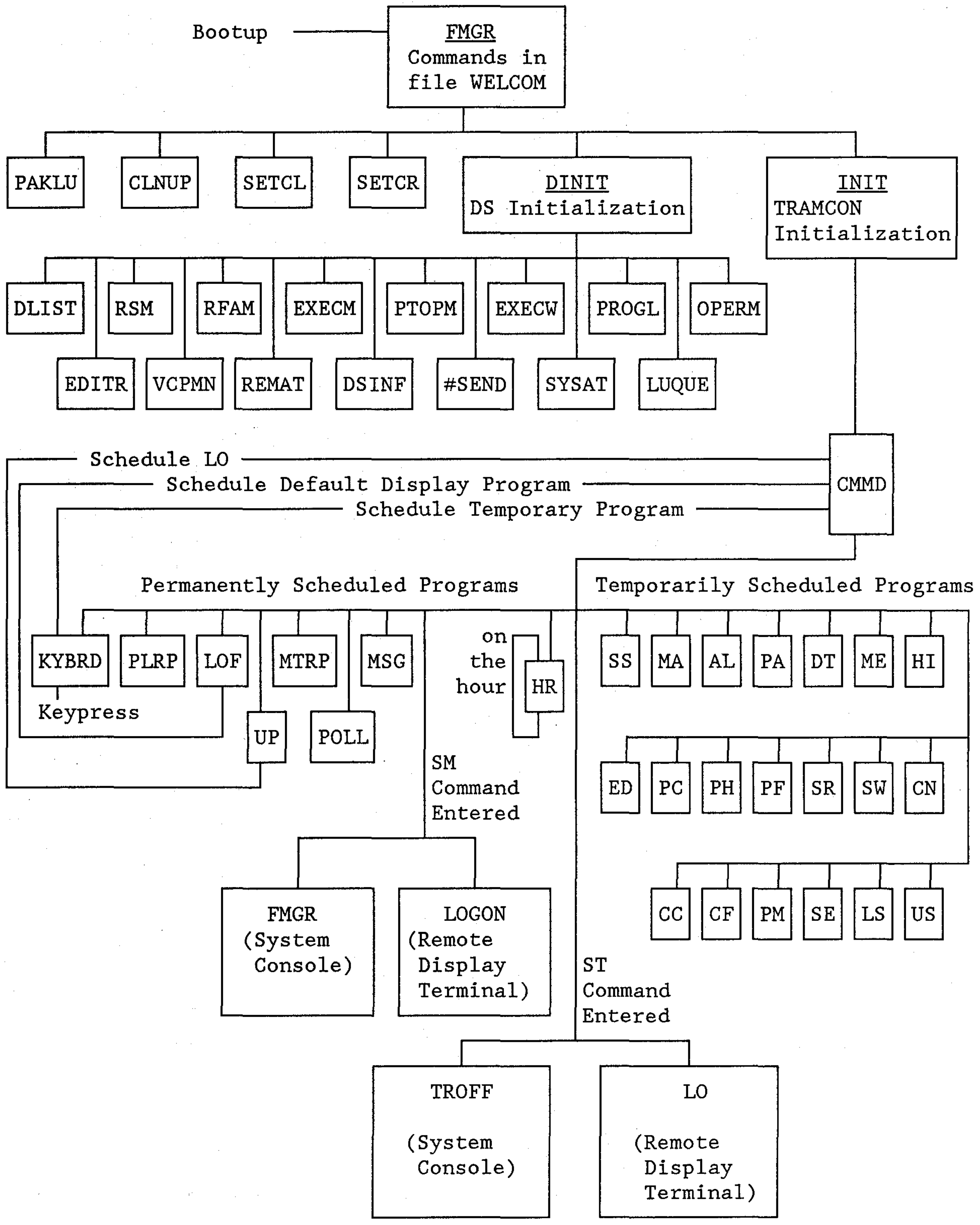


Figure 7. Program scheduling diagram.

It is desirable to be able to run several copies of many of these programs at the same time. For example, at three terminals on the same TRAMCON master, each operator may wish to schedule the SS display program at the same time. If only one copy of SS was available, two of the three terminal operators would have to wait while the SS display was being painted on the third terminal. This CLONING of programs is accomplished by composing a unique program name consisting of a two-letter command mnemonic concatenated with the two-character ASCII representation for the logical unit number corresponding to the terminal at which the command was entered. This "cloned" program is then executed by calling Routine "**schedule**" (alias EXEC) with function code 24. To further avoid running out of ID segments, before a new program is RESTORED (RP), old programs are removed from their ID segment by issuing the RTE "**OF**" command.

4.3 Run-string Parameters

Vital information is passed to each program as it is scheduled so that the programs can do such things as access the shared EMA and communicate with the appropriate terminal device. The RTE routine EXEC, called with function code 24, schedules programs and allows the caller to pass five one-word integer values to the program being scheduled. The scheduled program can recover these five values by calling the RTE routine RMPAR as the program begins to execute. Programs written in Pascal in the TRAMCON system recover these five one-word parameters by calling the routine "**get_parms**", which is an alias for the Pascal routine "**Pas.NumericParms**". An alias is used for "**Pas.NumericParms**" because identifiers cannot have a period in them in Pascal. The "**Pas.NumericParms**" routine is used instead of routine RMPAR because "run-time start-up code executed by all Pascal programs makes the use of RMPAR unreliable" (refer to Pascal/1000 Reference Manual, p. F-3).

Most programs need to access the large amount of data stored in a sharable partition of central memory. In the TRAMCON programs, this shared data area is referred to as the HEAP. The data stored in the HEAP are well structured in a hierarchical fashion. Therefore, to gain access to these data, a program must be given the address of the first word of data only. Given this single address, the newly scheduled program can determine any other address within the HEAP. The first problem is that this first word address cannot be passed directly in these one-word integer parameters because the HEAP, being larger than 32000 words, is referenced with two-word (32-bit) addresses.

NOTE

In HP PASCAL programs, this type of two-word addressable HEAP is referred to as HEAP 2. The HEAP 2 compiler OPTION appears in each program in the first line that begins with the PASCAL OPTION.

Therefore, the first run-time parameter, parm[1], is actually a CLASS number that has a two-word input buffer associated with it. To gain access to the HEAP, programs must call the RTE routine EXEC with a function code of 21

(CLASS GET) and reference the CLASS number passed in "**parms[1]**". Refer to Section 8.2.4 for details.

The last set of scheduling events deals with the orderly shutdown and subsequent start-up of TRAMCON software when executing the data base switchover command, CO. Refer to Section 14 for details on Configuration data base implementation.

5. TRAMCON COMMANDS

The operators of the TRAMCON system communicate with the TRAMCON computer by entering any one of the legal TRAMCON commands described in this section. The command is entered via the terminal keyboard. This section discusses the format of the operator commands and how the software maintainer can add or delete commands.

5.1 Command Format

This section describes the general format of the TRAMCON operator commands. Detailed descriptions of each command can be found in the TRAMCON Operator's Manual and on line using the ME command. To view the data being collected by the TRAMCON On-Line software, the operator must enter one of the legal TRAMCON commands listed in Figure 8. The commands are entered on a standard ASCII keyboard in response to the TRAMCON command prompt "**Enter Command:** ". The prompt is displayed by the TRAMCON module KYBRD. Refer to Section 9 of this manual for details of the TRAMCON command processing.

Each TRAMCON command is distinguished by its unique two-letter command code. This two-letter command code is the minimum operator input required for every TRAMCON command entry. Other information may be optional or required, depending on the particular command. Figure 8 lists the legal TRAMCON commands with their corresponding format, including the required portions and the optional portions.

The metalanguage used in Figure 8 to describe the format is interpreted as follows. The REQUIRED information in the command is NOT enclosed in brackets ([] or {}). The square brackets signify a single OPTIONAL field. If this field has further mutually exclusive options, then it is enclosed in the curly brackets with the enclosed options separated by the word "or". For example, "AL[,short_segment_name][,remote_id]{[,A] or [,P]}" says that the AL command requires the two letters "**AL**". The rest of the command, which is enclosed in brackets, is optional. The term "**remote_id**" indicates that a three-letter site code is accepted there. These site codes are unique for each location and are officially created and assigned by DCA. The optional field labeled "**short_segment_name**" stands for either a single digit "**segment ordinal**" or a six character "**short segment name**". These short segment names are defined by the Configuration Data Base Manager and must be spelled exactly as defined in the data base. The accepted spelling for these short segment names and their corresponding segment ordinals can be viewed on the TRAMCON terminal screen by entering the TRAMCON command, SE. The other optional fields are single letters (A, D, or P) and specify the general command directives ALL, DIAGNOSTIC, and PRINT. The curly brackets around "**[,A]** or **[,P]**" imply that the "**A**" and "**P**" options are accepted for the AL command, but only one or the other at any given time.

5.2 Command Parsing

The parsing of TRAMCON commands entered by an operator is done by the procedure "**parse_cmd**" in the central TRAMCON module CMMD and is described in detail in the CPCI documentation provided by ITS. The discussion in that document is more of a line-by-line English translation of the "**parse_cmd**" routine. This type of information is very useful and is NOT restated here. Instead, this document presents a discussion of the parsing process in more generic terms and discusses the changes or additional features that have occurred since the CPCI documents were written.

```
AC[,short_segment_name][,A][,remote_id]...[,remote_id]
AL[,short_segment_name][,remote_id][,P]
AR[,short_segment_name][,remote_id]
CC[,short_segment_name][,remote_id][,opposite_remote_id]
CN[short_segment_name][,remote_id][,S] or [,P]
CO
DE{[,SS] or [,MA]}
DI[,remote_id]
DT
ED
EN[,short_segment_name][,A][,remote_id]...[remote_id]
HE[,procedure_id][,P]
HI[,short_segment_name][,remote_id][,opposite_remote_id]
IN[,short_segment_name][,A][,remote_id]...[,remote_id]
LS[,short_segment_name][,remote_id][,P]
MA[,short_segment_name]
ME[,command][,CAT][,P]
OP,operator_id
PA[,short_segment_name]{[,remote_id]...[,remote_id] or [,A]}[,P]
PC[,short_segment_name]
PF[,P]
PH[,short_segment_name]
PM[,short_segment_name]
PO[,short_segment_name]{[,remote_id]...[,remote_id] or [,A]}
PR
SE[,P]
SR
SS[,short_segment_name]
ST
SW[,short_segment_name][,remote_id][,opposite_remote_id]
VE
WH
```

Figure 8. Legal unprotected TRAMCON commands and their syntax.

A working knowledge of the PASCAL set construct is required to fully understand how the command parsing works since the set construct is used extensively by the parsing code. The following sets of ASCII characters are defined globally in the program CMMD and are initialized at start-up in the routine "**Initialize**".

The set "**valid_chrs**" is initialized to include the characters 'A' through 'Z', '-', '+', '_', '&', '/' and '*'. The set "**digits**" is initialized to include the characters '0' through '9'. The set "**signs**" is initialized to include the characters '-' and '+'.

The command string is treated as a sequence of TOKENS, which are separated by commas. The parser accepts two general types of TOKENS. It accepts alphabetic tokens, those that begin with a character from the set (A - Z), and numeric tokens, those that are composed of characters from the set called "**digits**", which is defined above. The TOKEN gathering routine, "**nextok**", treats the comma as a command TOKEN delimiter. TOKENS are the "**words**" of the TRAMCON command language and the valid types of tokens are discussed in Section 5.1. Token length is limited to 20 characters, which is more than enough for all tokens currently in use. If a new TRAMCON command is introduced that requires a longer token, the appropriate arrays and loop limits must be increased in program CMMD. Since the comma is a token delimiter, it cannot be used as part of a token.

An exception to this general rule would be a token that is used exactly as entered. For example, in the OP command, the only token other than the command code OP is the actual operator name. Any key-press is allowed in the operator name and nothing is automatically capitalized. As this example shows, a token can be used exactly as it is entered because the procedure "**nextok**" collects TOKENS both exactly as entered and as legal TRAMCON command tokens. Therefore, if a future command requires a token with NO modifications, the value in "**as_is_tok**" can be used instead of the modified token value, which is placed into "**curtok**". The token gathering routine "**nextok**" modifies tokens, to produce the legal token value, as follows. All lowercase characters (a - z) are capitalized. Any character that is NOT in the set "**valid_chrs**" described above is ignored, except the comma, which is treated as the token delimiter as mentioned above. To adjust the legal operator key-presses, simply change the initialization statement for the set "**valid_chrs**" found in the procedure "**Initialize**" in program CMMD.

Figure 9 lists the hierarchy of token types and their allowable number of occurrences in any one command string.

TOKEN Type Hierarchy	Number of Occurrences
1. Command Code	1
2. Segment Name	1
3. Three-letter Site Code	21
4. Single-Letter Directive	1 each directive
5. Numeric	any number

Figure 9. TRAMCON command-type hierarchy.

The parser imposes the following order to the tokens in a given command string. The first token must be a one- or two-letter token and is interpreted as the command code. The first two alphabetic characters of the first legal token are interpreted as the command code and must match any two-letter entry in the command string literal "**cmd_alfa**", which is mentioned in Section 5.4. No other tokens are processed if the first legal token does NOT match a valid TRAMCON command.

NOTE

Single-character command codes are accepted by the parser ONLY when the master password is entered ("**access_restricted**" is false) and ONLY for selected single characters. This single character feature was originally introduced to reduce keyboard input requirements and increase the usability of the TRAMCON system.

After the command has been identified, the parser sets flags to determine what other token types to look for based on the particular command. This is done by setting the BOOLEAN flags "**check_segname**", "**check_scode**", "**check_link**", and "**check_print**" according to the contents of the appropriate set of commands. For example, if the AL command allows for the entry of a three-letter site code, then the set "**cmds_with_scode**" will include the "**al**" command. These sets are initialized in the routine "**Initialize**" in program CMMD. Adding a particular command to a set will cause the parser to look for that particular token type when processing the given command. These flags determine whether the parser looks for any occurrence of the given token type. If the flag is set, then the token type hierarchy list in Figure 9 is used by the parser to determine which type of token to attempt to match first.

The number of occurrences for each token is also important. As each token is identified according to the hierarchy listed in Figure 9, flags are set to indicate that certain token types have been found. This prevents the parser from attempting to turn every token into the token type at the top of the list. It also speeds up the parsing process by avoiding needless token processing if all the required parts of the command have been found. For example, if the AL command allows a segment name to be specified, each token received is first matched against the legal segment names. If the token does not match any segment name, is three characters long, and is NOT numeric, then it is interpreted as a three-letter site code.

Once a token is matched with a segment name, the flag "**seg_specified**" is set to true so that the parser does NOT attempt to interpret subsequent tokens as a segment name, since only one segment name is allowed per command. If the TOKEN does NOT match anything so far, a check is made for the single letter command directives, which are '**P**' for PRINT, '**A**' for ALL, and '**D**' for DIAGNOSTIC. The Diagnostic directive is allowed only if the master password has been entered (**heap^.access_restricted is false**). Refer to Section 12 for an explanation of the Diagnostic flag. The Print directive tells the computer to route the display output to the printer rather than the monitor. The ALL directive is a shorthand way of specifying ALL the remote units on a given segment.

All command qualifications such as "**system console only**" or "**poller only**" are implemented using sets of commands. To add or remove commands from any of these checks, you must change a line in the routine "**Initialize**" in program CMMD. For example, if you want to allow the SW command to be entered from any master regardless of Poller-Monitor status, simply remove "**sw**" from the statement in "**Initialize**" that assigns a value to the set "**poller_only**". Refer to Section 8 to implement the change.

After the input string has been successfully parsed, the parser does some preliminary syntax error checking. If a password is required, the parser prompts the operator for the password. If any syntax problems with the command are detected, the global variable "**cmd_err**" is set to the proper error number as shown in Figure 10.

No particular significance is attached to the order of the errors listed below. They are not in sequence because errors were created and deleted as the software was developed. In fact, the list shown in Figure 10 was extracted from the error message routine "**err_msg**".

Error message number 1 is displayed any time the parser does NOT match the Command Code entered or any time a command that requires the master password is entered and the master password has NOT been entered ("**restricted_access**" is true).

Error message number 5 is displayed any time the parser receives a command from a terminal other than the system console and that command is included in the set "**sys_console_only**".

Error Number	Error Description
1.	Command Unknown, check MENU
5.	This CRT NOT System Console
6.	Invalid CRT Location
7.	NOT Polling this segment
8.	Invalid Command Parameter
11.	No Link from xxxx to xxxx
12.	No remote unit specified
13.	Diagnostic in use
14.	No such remote on xxxxxx
16.	Printer NOT defined
17.	Printer NOT connected
19.	This CRT CAN'T be OFF-LINE

Figure 10. List of command errors produced by parser.

Error message number 7 is displayed whenever a command is received that is in the set "**poller_only**" and the given master is NOT in Poller mode on the given segment. An example of this type of command is the SW command. To avoid confusion developing from more than one master sending asynchronous messages on a given party-line poll channel, only the master that is exclusively in polling mode on the given segment is allowed to switch transmission equipments on the given segment. If this were NOT done, the result would be similar to allowing more than one master to send polling messages to remotes on a given segment. Most poll messages would collide on their way to the remote units, and any responses to the poll messages that did get through would also collide.

Error message number 8 is displayed when a given token cannot be matched with any valid token type.

Error message number 11 is displayed when the parser has found two site code tokens, but no link is found connecting these two sites on the current segment ("linkord" equals -2). To further aid the operator, if a link must be specified ("**link_required**" is true) and the operator specified only one site code, then the parser prompts the operator for the opposite end. The choices are deduced from the data base remote unit record. If there is only one possibility ("**max_link**" = 1), the software automatically chooses the opposite site and no prompting is done. If there are choices, the three-letter site codes for the opposite ends are displayed in the function key labels.

Error message number 12 is displayed when a site code is required ("**scode_required**" is true) by a particular command and none was entered ("**scode_entered**" is false).

Error message number 14 is similar to error 12, except in this case a site code was entered ("**scode_entered**" is true), but it did NOT match any site on the current segment ("**remotes_entered**" is empty).

Error message number 16 is displayed when the Print option ('P') is included in the command string and the configuration data for the given terminal indicates that this terminal does NOT have a printer attached (**printer_type=0**).

Lastly, error message number 17 is similar to error 16, except that the configuration data for the given terminal indicates that this terminal should have a printer attached. However, the returned value of the function "**printer_status**" is greater than zero, which implies that the actual status request sent to the terminal device indicated that a printer is NOT connected to the terminal (see appropriate Terminal Reference Manual, Device Status Request, <esc>&p<device code>^).

5.3 Command Entry Restrictions

This section discusses the various restrictions imposed by the parser on entry of individual commands and groups of commands. The parser enforces various restrictions on individual commands by checking for the inclusion of the given command in a set of commands which, as mentioned above, are initialized by program CMMD in routine "**Initialize**". The command restrictions currently enforced are listed in Figure 11.

All the sets listed above restrict the entry of their member commands by requiring certain environmental conditions to be true. The set "**poller_only**" requires that the given master be in poller mode for the current segment. The set "**sys_console_only**" requires that the command be entered from the system console. The set "**restricted_cmds**" requires that the master password flag, "**restricted_access**" be set to false. For example, any command included in the "**poller_only**" set will be accepted ONLY if the TRAMCON master at which the command was entered is in the polling mode for the current segment. Any command included in the "**sys_console_only**" set will NOT be accepted from any terminal on a given master except the system console. With these two sets we restrict the use of the SW command, for example, to one terminal on one master at any given time. The SW command is a member of both sets. Since the SW command is in "**poller_only**" and only one master can be Poller on a segment at one time, it is restricted to one master. Because the SW command is also in set "**sys_console_only**", it is further restricted to one terminal, namely, the system console, on that one master.

The "**restricted_cmds**" set includes all commands that will be allowed only if the "**restricted_access**" flag, which is found in the shared EMA HEAP, is set to false. Refer to Section 12 for a detailed discussion of the master password and the "**restricted_access**" flag. If "**restricted_access**" is true and a command in the set "**restricted_access**" is entered, the error message, "**Command Unknown, Check MENU**" is displayed at the terminal where the command was entered.

These restricted commands are NOT documented in the operator's manual, thus there is NO indication during normal operations that these restricted commands exist. Commands that are and should be included in this set are commands that are NOT vital to TRAMCON operation, but are useful for diagnosis, statistics gathering, and general troubleshooting. Again, refer to Section 12 of this manual for a detailed discussion of the current functions protected by the "**restricted_access**" flag.

SET Identifier	Current Value set by Initialize in CMMD
poller_only	[po,sw]
sys_console_only	[cf,co,dt,ed,lo,pm,po,pw,sc,sr,sw]
restricted_cmds	[cf,cr,dn,eq,lo,lu,ms,off,ru,sc,sm,up,us]

Figure 11. Command restriction SETS.

5.4 Adding, Changing, or Deleting TRAMCON Commands

The following is a list of valid TRAMCON commands:

1. **ac**	9. **de**	17. **hi**	25. **of**	33. **pr**	41. **ss**
2. **al**	10. **di**	18. **in**	26. **op**	34. **pw**	42. **st**
3. **ar**	11. **dn**	19. **lo**	27. **pa**	35. **ru**	43. **sw**
4. **cc**	12. **dt**	20. **ls**	28. **pc**	36. **sc**	44. **up**
5. **cf**	13. **ed**	21. **lu**	29. **pf**	37. **se**	45. **us**
6. **cn**	14. **en**	22. **ma**	30. **ph**	38. **si**	46. **ve**
7. **co**	15. **eq**	23. **me**	31. **pm**	39. **sm**	47. **wh**
8. **cr**	16. **he**	24. **ms**	32. **po**	40. **sr**	

The above list of commands is embodied in the software in the TYPE definition module [RECR3, approximately lines 98-100, with the class definition:

```
     cmds = (un,ma,ss,al,ar,pa,me,he,hi,cn,pc,ph,sw,cr,cc,cf,po,ac,
             ih,en,dt,pm,op,se,sm,si,de,pr,ls,sc,sr,ms,co,st,
             di,lo,wh,lu,eq,up,dn,off,ru,ve,us,pf,pw,ed,il).
```

The 47 legal commands listed above are bracketed by the two illegal commands "**un**" and "**il**", which stand for "**undefined**" and "**illegal.**" This allows the parsing routine, described in Section 5.1, to more easily scan the command class to determine if a given command is valid. Also, this allows commands to be added or deleted without having to change the code that checks these sets for command validity because the code loops are written to start with "**un**" and repeat until they get to "**il**". These checks are still valid as long as commands that are added or deleted are placed between these two bracketing commands. Each of the commands specified in "**cmds**" has a corresponding two-letter designator defined in the string constant definitions "**cmd_alfas1**" and "**cmd_alfas2**" in [RECR3 approximately lines 94 and 95:

```
     cmd_alfas1 = 'UNMASSALARPAMEHEHICNPCPHSWCRCCCFPOACINENDTPM';
     cmd_alfas2 = 'OPSESMSIDEPRLSSCSRMSOLCOSTDILOWHLUEQUPDNOFRUVEUSPFPWEDIL'.
```

Every two letters in the above two strings represent a valid operator entry. For clarity, whenever possible, the two letters used for the command identifier denoted in "**cmds**" above, are the same as the corresponding two-letter entry in the string literal "**cmd_alfas1**" and "**cmd_alfas2**" above. The only exceptions to this rule occur when the desired command mnemonic happens to be a reserved identifier in PASCAL such as "**in**" or "**of**". In these cases, the mnemonics are kept as desired, but the command identifers in the software are slightly altered to avoid conflict. For example, the RTE command to terminate a program is "**OF**", so the two-letter mnemonic is set to "**OF**", but the command identifier is changed to "**off**" so it will not conflict with the PASCAL reserved word "OF". It is not just coincidence that the command identifiers and the mnemonics appear to be in the same order. **That order must be maintained or the desired action will not result from a given command entry.** That is, the order in "**cmds**" must match the order in "**cmd_alfas1**" + "**cmd_alfas2**". The mnemonics are separated into two parts, "**cmd_alfas1**" and "**cmd_alfas2**", so that they would fit on an 80-column screen. They are logically the same string literal, and new commands can be placed into either one as long as their combined order matches that of the set "**cmds**".

NOTE

There is one more restriction concerning where a new command may be placed among the existing commands. This restriction is a result of the initialization code in routine "**Initialize**" in program CMMD, which copies the contents from these two-string literals into one local string array. Two **FOR** loops perform the transfer. The first loop indexes from "**un**" to "**pm**" and the other goes from "**op**" to "**il**". Therefore, either these loop indices must be changed, OR the new command must be added between "**un**" and "**pm**" or between "**op**" and "**il**".

To **ADD** a command, a new, unique two-letter designator must be placed in "**cmds**" and either "**cmd_alfas1**" or "**cmd_alfas2**", paying special attention to order. The command can be placed in any position between "**un**" and "**il**" (also, follow the NOTE above). To speed the command parsing, the commands used most often might be placed near the beginning.

To **DELETE** a command, just remove the identifier from "**cmds**" and the corresponding two-letter mnemonic from "**cmd_alfas1**" or "**cmd_alfas2**". After the new commands have been added or the old commands deleted or changed, the programs CMMD and US must be recompiled and relinked.

NOTE

When **DELETING** a command, all references to that command must also be removed. The only programs that have explicit references to individual commands are **CMMD** and **US**, with the majority of references occurring in **CMMD**. Most of the command references in program **CMMD** occur in routines **sched_dsp_prog**, **parse_it**, **process_simple_cmds** and Initialize.

6. REMOTE UNIT POLLING AND RESPONSE HANDLING

This section discusses how the TRAMCON remote units are polled by on-line software and how the software analyzes responses received from the remote units.

6.1 Remote Unit Polling

The main function of the TRAMCON system is to collect alarm/status indications from remote sensing devices (remote units), analyze the information, and present it to the user in a meaningful format. The communication between the master computers and the remote units is a serial asynchronous party line. The remote units currently used by TRAMCON respond only when they are asked to. A POLL message must be sent by the master computer asking for the alarm/status information known to a particular remote unit. Since the communication link is a party line, the POLL message must contain identification information that is recognizable by one, and only one, remote unit at a time. Because the particular make/model of remote unit currently used by TRAMCON is a non-computer-based dumb machine, no more information is necessary. Refer to Section 6.3 for a description of the POLL message.

Because of the party line arrangement, if more than one TRAMCON master is connected to the communication line, only one of those masters can be sending POLL messages to the remote units or POLL message collision would result. The Poller/Monitor flag "**poll_monitor**" indicates status for each segment on a given master.

NOTE

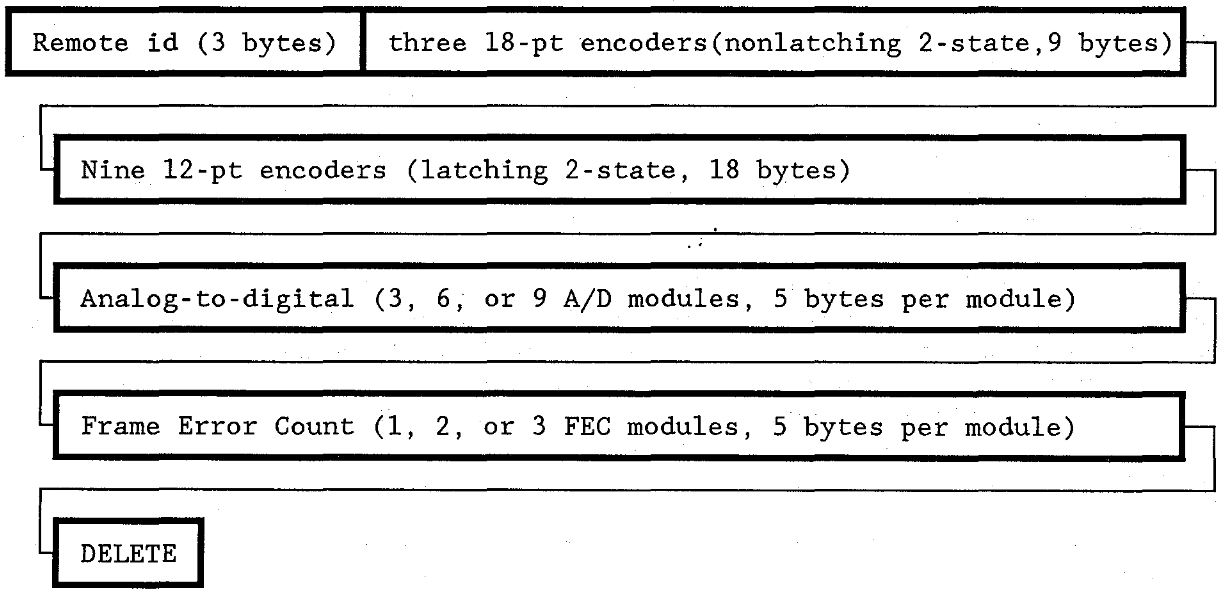
Although improper setting of these flags could cause devastating results, the coordination of the flags between masters remains a manual operation and the responsibility of TRAMCON operators. Operators must rely on phone communication with other master operators to confirm the status of other masters. Further, all the TRAMCON masters are connected in another communication network known as the IPC, which is fully capable of informing other masters of their Poller/Monitor status flags if not actually programmatically ensuring that the single Poller rule is obeyed. This capability exists but has NOT been implemented.

Further, POLL messages from independent, asynchronous modules on that one master must be presented to the communication channel in a serial fashion. To reiterate the main TRAMCON function is to collect alarm/status data, and that function can be kept serial by using a single module, namely PLRP, to issue POLL requests. There is a secondary function of TRAMCON, which requires that POLL messages be issued--the activation of relays at the remote units. This function is separate in all respects from that of the normal polling function. Therefore, this function is performed by another code module, namely, SW. Since these two modules run independently of each other and each module sends POLL messages over the same communication channel, a third code module was created to act as the POLL message "**clearing house**" and organizer. The name of this module is POLL. Using CLASS I/O, PLRP and SW send their POLL requests to the program POLL. Program POLL builds the actual polling message from information passed to it from either of those two programs. These messages are then sent over the communication channel to the remote units in the order they were received.

6.2 Physical Response to Logical Response Transformation

The TRAMCON system is composed of two main parts: the master computer and the remote units. The master computer receives responses from one or more remote units in a format dictated by the make and model of the remote unit. A remote unit with a new RAW response format NOT currently supported by TRAMCON constitutes a new remote unit type. The RAW response formats for currently supported DATALOK10 remote units (models 1D and 1E) are:

Normal RAW response for DATALOK10 Model 1D (50, 70, or 90 bytes)


Remote id (3 bytes)
three 18-pt encoders(nonlatching 2-state,9 bytes)
Nine 12-pt encoders (latching 2-state, 18 bytes)
Analog-to-digital (3, 6, or 9 A/D modules, 5 bytes per module)
Frame Error Count (1, 2, or 3 FEC modules, 5 bytes per module)
DELETE

Both response formats begin with the three-byte remote unit identifier and both end with the ASCII <DELETE> character.

The two-state alarm/status information is reported by the modules referred to as 18-point encoders and 12-point encoders. Each 18-pt encoder module reports 18 1-bit status indicators (6 bits per-message-byte). These status indicators represent the state of the equipment at the instant the remote unit was polled. They say nothing about the state of the equipment between polls.

Twelve alarm indicators are reported by each 12-pt encoder (6 bits per message byte). Unlike the 18-pt encoder data, these indicators are latching. That is, if a 12-pt indicator is set ON at any time between polls, that indicator stays ON until the remote unit is polled. The number of 18-pt and 12-pt encoder modules installed in each model of the DATALOK10 is fixed. Therefore, the number of alarm/status indicators reported does not vary regardless of how many sets of communications equipment are being monitored by the given remote unit.

Notice, however that both the 1D and the 1E response diagrams show three valid response lengths. The model 1D response can be 50, 70, or 90 bytes long, and the model 1D response can be 129, 145, or 161 bytes long. This variable length results from the fact that each unit is defined to be able to monitor up to three link ends (three sets of radio equipment). A DATALOK10 model 1D remote unit that is wired to one set of radio equipment will respond with a 50-byte message (NOT including the DELETE character). Since the response length is variable, the <DELETE> character is used by the poll channel driver (DVA76, see Section 6.3) to determine the end of the response.

This variable response length is also indicated by a variation in the amount of hardware modules installed in any given unit. The variant modules are installed at the end of the unit and, therefore, report their information at the end of the response. In each model, the number of Frame Error Counter (FEC) modules is variable depending, again, on the number of sets of radio equipment being monitored by that unit. Also, in the older model 1D, each A/D value is reported by a single A/D module. Therefore, the number of A/D modules in the 1D model is variable. The diagram for the model 1D response indicates that there can be 3, 6, or 9 A/D modules and 1, 2, or 3 FEC modules installed, depending upon whether the remote unit is monitoring 1, 2, or 3 sets of communications equipment.

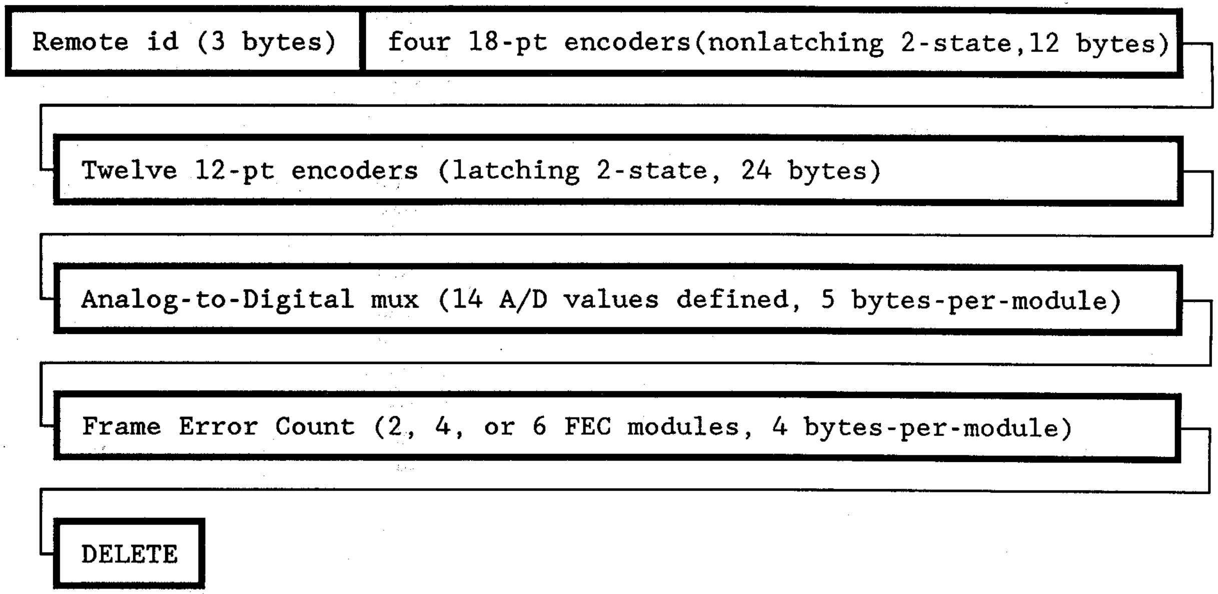
The A/D module for the model 1E is referred to as a MUX card because it can report up to 16 A/D values from one module. The model 1E always has one A/D MUX module installed, regardless of how many sets of communications equipment it is monitoring. The number of A/D values that are actually reported by the A/D module are specified by two additional bytes (A/D card select and A/D mode/point) in the polling message sent by the master (see Section 6.3). The TRAMCON software program POLL currently requests 14 of the 16 possible values from the remote unit by setting the fourth and fifth bytes of the poll message to the constants "**a2d_card_select**" and "**a2d_nbr_values**". These two constants are defined in INCLUDE module [RECR3. Each A/D value reported adds five bytes to the response.

Each FEC module reports two binary-coded-decimal (BCD) values in four bytes of the response. The model 1D FEC modules also report a 1-byte identifier, since the A/D information preceding the FEC data varies in length. For the software to interpret the FEC data from the model 1D properly, strap 13 on each FEC module must be set to "**A**" causing the FEC card to report an identifier byte. Further, the first FEC module must have the S3 rocker switches set to 001110. Software routine "**unpack_response**" in $MPLIB checks for the ASCII character "**c**", which corresponds to the S3 setting just mentioned, to determine the end of the A/D data and the start of the FEC data.

One of the prime TRAMCON design goals was to make the TRAMCON software able to support any new remote unit type with a minimum amount of code change or code addition. To accomplish this goal, most of the remote unit response processing code was written to process a generic response format, defined in Section 11.3. The GENERIC response is defined by the RECORD type "**unpacked_response**" found in INCLUDE module [RECR3 (refer to Section 11.1.2).

The majority of the code does not have to be adjusted each time a new remote unit type is added to the list of remote units supported by TRAMCON. In other words, each unique response received from a given physical remote unit is immediatel, translated (by routines "**unpack_response**" and "**transform_ordinal**" in $MPLIB) into the same generic response format as any other response. Therefore, the only additional code required to support another physically different remote unit is a few lines in the transformation routine that translate the uniquely formatted response into the generic response format. Once this transformation is complete, the new response looks just like any other response from any other remote unit.

Normal RAW response for DATALOK10 Model 1E (129, 145, or 161 bytes)



Remote id (3 bytes)
four 18-pt encoders(nonlatching 2-state,12 bytes)
Twelve 12-pt encoders (latching 2-state, 24 bytes)
Analog-to-Digital mux (14 A/D values defined, 5 bytes-per-module)
Frame Error Count (2, 4, or 6 FEC modules, 4 bytes-per-module)
DELETE

The routines that must be modified to accommodate a new remote unit are "**transform_ordinal**", "**unpack_response**", and "**print_response**" in $MPLIB and a few lines in the program SW to handle the new relay switch assignments.

The TRAMCON On-Line software currently supports two models of the Pulsecom DATALOK10 remote unit. Even though the models 1D and 1E are both DATALOK10 remote units, their responses are physically different and therefore they are seen by TRAMCON as different remote unit types.

6.3 The POLL/RESPONSE Interface Driver - DVA76

The device driver, DVA76, is used to handle I/O between a TRAMCON master and the Pulsecom DATALOK10 remote units over an RS-232 serial port using a HP BACI hardware interface plugged into port 15 (octal) of the TRAMCON master computer's backplane. This driver is a modification of the HP terminal driver, DVA05, and is written in HP-1000 macro language and standard HP-1000 driver format.

There are two entry points into the driver:

(1) IA76, the initialization entry point. Each time an I/O request is started by the operating system on channel 15, the driver is entered through the initialization entry point.

(2) CA76, the continuation/completion entry point. Each time the operating system returns to the driver to continue or complete an I/O request, the continuation/completion entry point is used.

There are only two I/O requests processed by module DVA76: (1) a write request, which is a TRAMCON remote unit polling request or a remote unit relay switch request, and (2) a read request, which is a TRAMCON master request to receive a response transmitted by a remote unit. If a TRAMCON master is in MONITOR mode for a particular segment, only read requests will be issued to DVA76 by the program MTRP. If the master is in POLLER mode for a given segment, the program PLRP will issue write requests followed immediately by read requests to accept the response from the remote unit for which the write (poll or relay switch) request was issued. Write requests are also issued by the program SW in response to an operator request to activate a relay. Both PLRP and SW do not actually issue the write request directly to the driver. Instead, to ensure that all requests issued on a given channel are kept in sequence, the originating programs such as PLRP and SW send their requests to a central request handler, POLL, that actually issues the write request to the driver. POLL's request has the format:

Normal poll request for DATALOK10 Model 1D (4 bytes)

Remote id (3 bytes)	DELETE

Normal poll request for DATALOK10 Model 1E (6 bytes)

Remote id (3 bytes)	A/D Card Sel	A/D Mode/Pt	DELETE

The TRAMCON software program POLL currently requests 14 of the 16 possible A/D values from the model 1E remote unit by setting the fourth and fifth bytes of the poll message to the constants "**a2d_card_select**" and "**a2d_nbr_values**". These two constants are defined in INCLUDE module [RECR3.

Relay Switch request (6 bytes)

Remote id (3 bytes)	card select	relay select	DELETE

The read request is issued to the driver either by program POLL immediately after the write request, or by program MTRP when in MONITOR mode on a given segment. Processing of the read request starts at label LISTN. The caller's buffer pointers are initialized, and the BACI interface is set to receive mode by issuing a master RESET. The BACI is also set to CHARACTER mode so that an interrupt will be generated when any character is received. The driver is exited with JSB EXIT1 (approximately line 299) and will resume at this position if a character is received before time out. There are two values for this wait-for-response time out, a 4-second time out for POLLER mode where the read is issued immediately after the write and a 1-minute timeout for a read request issued by MTRP. When in POLLER mode, the longest possible response from the currently used DATALOK10 is 6.5 seconds. Coupled with the fact that a DATALOK10 responds immediately after receiving a poll request, this allows us to know that if a complete response is not received within approximately 7 seconds, the given remote unit is not responding properly. On the other hand, if the given master is simply monitoring the given segment for any response, a much greater time must be allowed before giving up waiting for a response. If a character is received by the BACI before this read timeout, it is assumed that a remote response is coming in, and processing resumes just after the JSB EXIT1 (approximately line 300). Once a response has started arriving, it will continue without interruption at 300 baud until the DELETE character is received. Since the transmission speed is slow relative to the rate at which the characters can be removed from the BACI buffer and transferred to the caller's buffer, the BACI buffer will periodically empty, causing the BACI to generate a BUFFER EMPTY interrupt. If so, the driver is exited with the return address set to label CKINT.

These two requests are the only TRAMCON master-to-remote unit communication requests processed by DVA76. The write request is TRAMCON master-to-remote unit communication and the read is from remote unit to TRAMCON master. The write request is completed by the driver when an ASCII DELETE character is transmitted to the remote unit. The read request is terminated/completed by the driver when either an ASCII DELETE character is received from the remote unit or the request times out. The driver is set to handle its own interrupts. That is, when the operating system receives an interrupt on channel 15, it branches into the driver, DVA76, to process the interrupt. The only interrupts processed by DVA76 on a read request are Special Character (DELETE), BACI Buffer Half Full, BACI Buffer Full, BACI Buffer Empty, and Parity Error. All other interrupts are treated as spurious and ignored.

Other than the two I/O requests, there are a few BACI interface configuration/control requests that are processed by DVA76. The following parameters can be programmatically set on the BACI interface:

Baud rate	**- 110 to 9600 bps**
STOP bits	**- 1 or 2**
Parity	**- none, odd, or even**
Data bits	**- 7 or 8**

Since the Pulsecom DATALOK10 is a hard-wired, non-programmable device, the BACI configuration must be done only after any power failure instead of before each I/O request. For the DATALOK10 remote unit, the BACI interface is configured by program CMMD as follows:

Baud Rate: 300 bps, STOP bits = 2, parity = even, data bits = 7

Another control request sets the read request time out value, which was chosen to be 1 second until receipt of the first response byte after a write (polling) request has been issued and 1 minute for any response from any remote unit on channel 15 (octal) if no polling requests are being sent for the given segment on the given TRAMCON master (master is in MONITOR mode for the given segment).

The TRAMCON master channel over which DVA76 communicates is hard-coded by the statement "**CARD EQT 13**" to be slot 13 (15 OCTAL). The only difference between DVA76 and DVA77 is the hard-coded channel number which is 13 (15 OCTAL) for DVA76 and 14 (16 OCTAL) for DVA77. Other than that, the above description applies, word for word, to driver DVA77.

6.4 PHYSICAL vs LOGICAL Remote Unit

A single physical TRAMCON remote unit (currently the Pulsecom DATALOK10) was defined to be able to monitor up to three sets of communications equipment plus the SITE equipment. This limit of three sets of communications equipment per remote unit is a result of the number of alarm/status points available on the DATALOK10 Model 1E and the number of alarm/status indicators to be monitored on the DRAMA radio equipment, which was the first set of equipment to be monitored by TRAMCON. A margin of about 10% was built in for additional alarms or support of new equipment with more alarm/status points than the DRAMA system.

This three-way limitation for a single remote unit on a given segment proves to be quite adequate for most locations monitored by TRAMCON. Early on in the TRAMCON implementation, a few locations had more than three sets of communications equipment that needed to be monitored by a single remote unit on a given segment. These requirements do not seem to be isolated cases and additional similiar situations are anticipated.

The solution to this problem was to introduce the concept of a LOGICAL remote unit. A LOGICAL remote unit consists of one or more PHYSICAL remote units. The TRAMCON operator interacts with LOGICAL remote units while the On-Line software continues to process responses from PHYSICAL remote units. By continuing to process the PHYSICAL responses individually, a minimum of software change was necessary to implement the greater than three-way remote unit. Both configuration data and run-time data, kept the same three-way definition. PHYSICAL remote units can now be associated with one another via a linked list to comprise a LOGICAL remote unit. The linked list is implemented with the two fields "**extent_of**" and "**next_extent**" in the run-time Remote record "**remote_status_record**".

These list pointers are initialized by the program INIT when the TRAMCON software is booted up. A few simple assumptions are made by INIT when setting these pointers. First, all remote unit records on a given segment that have the same SITE record pointer are considered to be components of a LOGICAL remote unit and are linked together. Note that these remote units do NOT have to be contiguous. When more than one record with the same SITE pointer is found, the first record defined is considered to be the MAIN component. The MAIN component of a MULTIPLE Remote is used by the On-Line software to supply the SITE alarm/status information. The MAIN component is marked by setting the pointer "**extent_of**" to -1. The field "**next_extent**", in all components including the MAIN component, is given a value greater than -1 if this is not the last component of the MULTIPLE remote unit. For most remote units that are single units, both "**extent_of**" and "**next_extent**" are set to -1.

The value of these pointers is actually an index into the array "**remote_info**" in the "**segment_record**". For example, assume that on segment DEB4C the remote units pointed to by "**remote_info**[2]" and "**remote_info**[3]" both have SITE record pointers that point to the same SITE record ANU. The MAIN component will be the first one encountered, namely the record pointed to in "**remote_info**[2]". In the corresponding run-time data record "**remote_status**[2]^", the field "**extent_of**" would be set to -1 and the field "**next_extent**" would be set to 3 pointing to the next component of the MULTIPLE remote unit. For the second component, "**remote_status**[3]^", field "**extent_of**" would be set to 2 indicating that this component is NOT the MAIN component and is an extension of the MAIN component in "**remote_info**[2]". The field "**next_extent**" would be set to -1 to indicate that there are no more components of this MULTIPLE remote unit.

NOTE

Which communications equipment is connected to which physical Remote Unit is transparent to the TRAMCON operator since all categories of data are presented to the operator as if they are being monitored by a single remote unit. On the other hand, this is of great concern for installation personnel and the configuration data base maintenance personnel. Both of these groups must be aware of the assumptions made by the On-Line software mentioned above.
First, the SITE category must be wired to the MAIN or first unit defined in the array "**remote_info**". Second, the SITE record pointer for all components must be exactly the same. The On-Line software does not require that all categories be defined in one unit before a category can be defined in the next component unit. But close coordination must still be maintained between the installation drawing team and the data base designer so that the the data base Remote record definitions exactly match the drawings.

All programs that formerly dealt with single physical remote units must now include the procedures in the module [EXTNT. This module will present the response data from any number of physical remote units to the operator as if it is the data from one remote unit. The module [EXTNT contains CONST, TYPE, and VAR sections followed by two procedures, and must be included just before any other procedures declared in the program. The best location for most TRAMCON programs is immediately after the $INCLUDE "[**TRVAR**"$ statement.

When displaying the alarm/status information for a selected remote unit, programs such as AL must not only search for all categories defined in one "**remote_record**", but they must now also follow the chain indicated by the value "**next_extent**". In the past, when the defined categories for a given "**remote_record**" were displayed, the entire remote units response was displayed. The entire response is now displayed only when all defined categories are displayed and the "**next_extent**" pointer equals -1.

7. MAINTAINING THE MENU AND HELP TEXT FILES

Two information facilities have been developed to provide the TRAMCON user with On-Line help in using the TRAMCON system and performing other site specific functions. These two aids are referred to as MENU and HELP. Both aids are similar in structure and operation and differ only in the type of assistance they provide.

First, the MENU provides the operator with On-Line text descriptions of the TRAMCON commands. These descriptions can be displayed or hard copied by entering the TRAMCON command ME. Theoretically, the only TRAMCON command that the new operator must be informed of is the ME command, since entering this command will lead the operator to descriptions of ALL TRAMCON commands including ME.

The HE command produces similar text results, not for TRAMCON commands, but for procedures related to the operation of TRAMCON or any other operational aspect of the particular site. This file can and should be altered by each site to reflect the site's particular way of performing its functions.

The data are stored as text information on type 4 disc files (variable-record-length TEXT). Even though these two commands deal with different data, the storing and maintenance of the text files is identical. First, the data are divided into two levels. The first level is the list of all commands/procedures that are currently described in the text files. This list contains a one-line entry for each command/procedure. This one line must have the two-letter command/procedure identifier as the first two characters in the line, followed by any phrase that briefly describes the command/procedure. The two-letter identifier must be unique within both sets of information.

The menu (TRAMCON command) list has one other aspect that the HELP (procedures) list does not have. That is, the commands are grouped into categories or sets of commands that perform similar functions. The categories are indicated by a one-line entry similar in format to the command entries described above with the two-letter identifier set to XX. Therefore, there can be no XX command in TRAMCON. All commands following a given category, up to the next category, are included in the given category. The second level is a detailed description for each command/procedure. These descriptions have no particular format, but are simply text that might aid in the use or understanding of a particular command or procedure. The text data described above are stored in the following type 4 files on disc, Logical Unit 10.

"CM1 - This file contains a one-line title for each TRAMCON command. If any command is added or deleted, the appropriate line must be added or deleted from this file using the TEXT file editor EDIT.

"CM - This file contains the detailed descriptions of the TRAMCON commands. The command descriptions within the file are separated within the text file by a line of text consisting of the two characters "" as the first (and usually the only) two characters in the line. Figure 12 shows the entry for the ME command.

```
ME[,command][,CAT][,P]
This command is used to provide a list of operator commands and to
provide information on the use of each of the operator commands. If the
command "ME" is entered alone, the list of commands in alphabetical order
with one-line descriptions will be shown on the screen. If the entry is
followed by any of the other two-letter commands, a descriptive paragraph
concerning the command and its entry syntax will be brought to the
screen. If either of the preceding entries is made followed by a "P",
the menu or the descriptive paragraph will be printed out. If the
command "ME,CAT" is entered, the menu of commands will be listed on the
screen by category and if this entry is followed by ",P", the menu will
be printed in this order.
```

Figure 12. Sample entry in file "CM.

Figure 12 shows that the first line of each entry should be a sample of the specific command format written in the metalanguage discussed in Section 5.1 of this manual. To maintain this file, the operator simply runs the HP text editor program EDIT, which is provided with the TRAMCON system.

"HE - This file contains the detailed procedure descriptions that are referenced by the HE command. A sample of the data in this file is shown in Figure 13.

Just like the "CM file above, the first two characters of each line in file "HE are the keys to the rest of the line. The two characters "" are used to separate one procedure description from another. The first line of each description should be a phrase describing the procedure. This line should begin with the two-letter procedure identifier followed by a descriptive phrase that starts in column 11. For example, the first line of the entry for the BO procedure shown in Figure 13 reads as follows:

"Bootup   System bootup procedure".

Unlike the command file, this first line does affect what the HE command displays on the screen for the Procedures menu. The operator can alter the text information for file "HE using the TRAMCON command ED. When the operator is finished altering the information, the program ED automatically updates the index and title information as discussed below.

```
                              .
                              .
                              .
Fill this in later with Tech control SOP
""
BOot-up   System boot-up procedure
The procedure given here is used to re-start the computer if after a
repair or other problem situation the "RUN" light on the computer is
not lighted.
  PROCEDURAL STEPS
  1. Press the computer "HALT" button.
  2. Select the "S" register.
  3. Set bits 15, 14, 12, 9, 7, 1 (151202 octal) ON in the "S" register.
  4. Press "STORE".
  5. Press "PRESET".
  6. Press "IBL".
  7. Press "PRESET" (again).
  8. Press "RUN".
This procedure will start the computer if the proper programs are
loaded. The disc memory will rattle a bit, the default display will
be shown on the terminal and TRAMCON will be running. If this result
is not observed, it may be necessary to reload the system software
tape following the "TAPE LOAD" help procedure.  If this does not
correct the problem, follow the "FAilure" procedure to have
the computer restored to service.
""
FAilure   Restoring the TRAMCON master to service
In the event that the master computer cannot be brought up using the
                              .
                              .
                              .
```

Figure 13. Sample entry in file "HE.

The following discussion encompasses for both the TRAMCON command and the procedure information.

As the text files shown above grow, especially the procedure file, the time needed to search for a selected entry increases. To speed up this search, the text files were indexed and the indexes stored in a type 2 (fixed record length, random access) file on disc. Since the text files are type 3 or 4, the addresses of the individual records within the file will vary as the information is corrected, changed, or added. Whenever the text information described above is altered, the corresponding index file must be updated. The index files for the command and procedure indices are named "CMIDX and "HEIDX respectively. The record definition for these files is shown in Figure 14.

```
me_index_record = ARRAY[1..max_idx] OF  RECORD
                                        idx_key: two_chars;
                                        title: text_line_type;
                                        titlelen, recnbr, block, wrd: INT
                                        END;
```

Figure 14. Index file record definition.

The first thing to notice when studying the record definition in Figure 14 is that there is more than just the record address stored here. The record address is a physical disc address and consists of the three one-word values "**recnbr**", "**block**" and "**wrd**". These three values represent a disc address as explained in the HP Programer's Reference Manual, pp. 3-61, LOCF Calls. The "**idx_key**" is the two-letter value used to uniquely identify each command or procedure mentioned above. The "title" field is an ASCII character string, 80 characters long, that is a one-line description of the given command/procedure. The maximum length for the title is 80 characters, but the title can actually be any length up to 80. Therefore, for display purposes, the actual title length, in characters, is stored in the variable "**titlelen**".

"CMIDX - This file contains the index records for all the currently defined TRAMCON commands. There is one record for each command. Figure 14 describes the index records. The information contained in these records is updated by the program MEIDX. Each time the contents of file "CM is altered by software maintenance people using the program EDIT, the program MEIDX must be executed to place the new pointer values into this file. To execute program MEIDX, the operator should be in FMGR or Session Monitor. At the colon prompt, the operator enters "**MEIDX <RETURN>**". MEIDX automatically updates the values "**recnbr**", "**block**", and "**wrd**" to correspond to the actual record addresses in text file "CM described above.

"MEIDX - This file contains the same information as described above for file "CMIDX except that the information in this file applies to the procedure function instead of the TRAMCON command function. Also, updating of data in this index file is done automatically by the program ED after the TRAMCON operator has changed the contents of file "HE using the TRAMCON command ED. Unlike the TRAMCON command function, the index and title information is automatically updated by the program ED instead of requiring the operator to manually update this information by running MEIDX.

For each command entry in file "CMIDX, the actual physical disc address of the first word of the corresponding command description in file "CM is computed by calling the FMP routine LOCF. In the text file there are many disc records, but only the addresses of the first word of each command description is desired. To locate these first words, MEIDX reads text file "CM until the command description separator "" is found. The record following this is the first word of the next description. It is easy to see that there is dependance on order between the two files "CM and "CMIDX. That is, the first command description found in file "CM must correspond to the first entry in the index array in file "CMIDX and so on.

Deleting a command requires that the description be removed from "CM and, at the same time, the index for the given command must be removed from the array in file "CMIDX by moving all entries following the deleted entry forward one location.

Updating of the commands function is not as automatic as that for the procedures. This must be accomplished manually by software maintenance personnel as follows. First, the updates must be made to the two text files "CM and "CM1 using the program EDIT being careful to maintain order and a one-to-one correspondence between these two files. Second, the command titles must be extracted from file "CM1 and placed in index file "CMIDX by executing the program MEDX1. Last, the disc addresses must be updated in the index file by executing MEIDX. The steps necessary to implement a change to the TRAMCON command descriptions are summarized in Figure 15.

When the operator is finished changing the TEXT descriptions in file "HE, the program ED updates the information in file "HEIDX as follows. Each line of text is read from file "HE. If this is the first line of a given procedure entry, the first two characters are assumed to be the procedure identifier and are stored in the index record variable "**idx_key**". The disc address of the first word of this first line is determined by a call to the FMP procedure LOCF and stored in the index record in variables "**recnbr**", "**block**" and "**wrd**". Starting with character position 11, the rest of the first line is assumed to be the descriptive phrase for this procedure and is stored in the index record variable "**title**". The length of the title is stored in the index record variable "**titlelen**". Therefore, even though the TRAMCON operator does not explicitly update the index record for these procedures, the information that goes into the index is directly dependent on the information entered by the operator in line one of the text description. Also, the format of that first line is critical. The first two character positions determine the procedure identifier. Positions 3 through 10 are ignored and the rest is used as the procedure title. The identifier and the title are used by program ME to display the list of procedures that are defined. They are also displayed as the first line by ME when displaying the procedure description for a particular procedure as shown in Figure 13.

1.	EDIT,"CM	Change command description
2.	EDIT,"CM1	Change command two-letter ID and/or title
3.	MEDX1	Update titles in File "CMIDX
4.	MEIDX	Update disc addresses in File "CMIDX

Figure 15. Steps to change/add/delete TRAMCON command descriptions.

8. SOFTWARE DEVELOPMENT AND MAINTENANCE

Section 8.1 enumerates the eight steps necessary to develop and implement the TRAMCON software.

The remainder of this section discusses how the tools listed below are used to develop and maintain the TRAMCON software.

8.1 Software Development and Maintenance Tools

The eight steps involved in software development and maintenance along with the software modules used to accomplish each step are listed below.

```
1.  Operating System Generation/Configuration
     RT6GN - Operating system generator
             RTE-6/VM On-Line Generator Reference Manual, Part No.92084-90010
     SWTCH - Program to implement a newly generated operating system
             RTE-6/VM System Manager's Reference Manual, Chapter 5
             Part No. 92084-90009

2.  Source Code Creation and Editing
     EDIT  - The Source Code Editor
             EDIT/1000 User's Manual, Part No. 92074-90001

3.  Compiling/Assembling
     PASCL - Pascal compiler
             Pascal/1000 Reference Manual, Part No. 92833-90001
     FTN7X - Fortran compiler
             Fortran 77 Reference Manual, Part No. 92836-90001
     MACRO - HP-1000 assembler
             Macro/1000 Reference Manual, Part No. 92059-90001
     SXREF - Assembler Cross Reference
             Macro/1000 Reference Manual, Part No. 92059-90001, Appendix F

4.  Segmenting (for large programs)
     SGMTR - Large program segmenter
             RTE-6/VM Loader Reference Manual, Chapter 6, Part No.92084-90008
     INDXR - Creates Indexed-Merged files for the Segmenter and Loader
             RTE-6/VM Loader Reference Manual, Part No.92084-90008, p.6-41

5.  Indexing (for libraries)
     LINDX - Library indexer

6.  Linking/Loading
     LINK  - Program linker
             RTE-6/VM LINK User's Manual, Part No. 92084-90038
     LOADR - Program loader
             RTE-6/VM Loader Reference Manual, Part No. 92084-90008
     MLLDR - Loader for large segmented programs
             RTE-6/VM Loader Reference Manual, Part No. 92084-90008
```

7. File Creation, Backup, and Recovery

FC	- File Copy Utility RTE-6/VM Utility Programs Reference Manual, Chapter 4 Part No. 92084-90007

8. Debug, Status, Troubleshooting, Utility

WHZAT	- Snapshot of Program Activity RTE-6/VM Utility Programs Reference Manual, Chapter 2 Part No. 92084-90007
LGTAT	- Log Track-Assignment Table Utility RTE-6/VM Utility Programs Reference Manual, Chapter 2 Part No. 92084-90007
LUPRN	- System Configuration Utility . RTE-6/VM Utility Programs Reference Manual, Chapter 2 Part No. 92084-90007
DLX	- FMGR Directory Utility, from HP-1000 User's Group
LST	- TEXT File Listing Utility, written by ITS
CLASS	- CLASS I/O Information Utility, from HP-1000 User's Group
SAM	- System Available Memory Status Utility, from HP-1000 User's Group

8.2 Software Development and Maintenance Procedures

The following figures are lists of procedure files that were used to develop the TRAMCON On-Line software. These procedure files should be executed in total to redo the entire TRAMCON On-Line software system, or, should be consulted to discover how each individual module might be redone (compiled, segmented, indexed, and linked, or loaded). The procedure files for the compiling and loading of data base Configurator program CONFI are listed in Appendix A of this manual.

8.2.1 Editing

The editing of software modules is done using the text file editor EDIT supplied by HP. The editor creates and maintains type 4 text files on disc. The file naming conventions are:

(1) All program SOURCE file names begin with the letter "&", followed by the executable program name. (e.g., program AL has source file name &AL)
NOTE: Most program names are two letters long and match the corresponding two-letter TRAMCON command mnemonic.

(2) All INCLUDE modules begin with the letter "["

See Section 8.4 for a discussion of the structure of these source files.

8.2.2 Compiling and Assembling

Figure 16 is the procedure for compiling and assembling all of the TRAMCON On-Line software.

```
:SV,4
:RU,P,&AL,,%AL::10
:RU,P,&ARPTR,,%ARPTR::10
:RU,P,&BROAD,,%BROAD::10
:RU,P,&CC,,%CC::10
:RU,P,&CF,,%CF::10
:RU,P,&CHECK,,%CHECK::10
:RU,P,&CMMD,,%CMMD::10
:RU,P,&CN,,%CN::10
:RU,P,&CO,,%CO::10
:RU,P,&CR,,%CR::10
:RU,MACRO,&CRSET,,-
:RU,MACRO,&DSNRV,,-
:RU,P,&DT,,%DT::10
:RU,MACRO,&DVA76,,-
:RU,MACRO,&DVA77,,-
:RU,P,&ED,,%ED::10
:RU,MACRO,&GTTIM,,-
:RU,P,&HI,,%HI::10
:RU,P,&HR,,%HR::10
:RU,P,&INIT,,%INIT::10
:RU,MACRO,&JULIN,,-
:RU,P,&KYBRD,,%KYBRD::10
:RU,P,&LO,,%LO::10
:RU,P,&LOF,,%LOF::10
:RU,P,&LON,,%LON::10
:RU,P,&LS,,%LS::10
:RU,P,&MA,,%MA::10
:RU,P,&ME,,%ME::10
:RU,P,&MEDX1,,%MEDX1::10
:RU,P,&MEIDX,,%MEIDX::10
:RU,P,&MPLIB,,%MPLIB::10
:RU,LINDX,%MPLIB,$X::10
:PU,$MPLIB
:ST,$X,$MPLIB::10:5:-1
:PU,$X
:RU,P,&MS,,%MS::10
:RU,P,&MSG,,%MSG::10
:RU,P,&MTRP,,%MTRP::10
:RU,P,&PA,,%PA::10
:RU,P,&PC,,%PC::10
:RU,P,&PF,,%PF::10
```

Figure 16. Procedure file for compiling TRAMCON On-Line software modules.

```
:RU,P,&PH,,%PH::10
:RU,P,&PLRP,,%PLRP::10
:RU,P,&PM,,%PM::10
:RU,P,&POLL,,%POLL::10
:RU,P,&PR,,%PR::10
:RU,P,&RMAST,,%RMAST::10
:RU,P,&SC,,%SC::10
:RU,P,&SE,,%SE::10
:RU,P,&SETCL,,%SETCL::10
:RU,P,&SETCR,,%SETCR::10
:RU,P,&SETDT,,%SETDT::10
:RU,P,&SETVE,,%SETVE::10
:RU,P,&SI,,%SI::10
:RU,P,&SR,,%SR::10
:RU,P,&SS,,%SS::10
:RU,P,&SW,,%SW::10
:RU,MACRO,&T2,,-
:RU,P,&TIMPA,,%TIMPA::10
:RU,P,&TIMSE,,%TIMSE::10
:RU,P,&TRLIB,,%TRLIB::10
:RU,LINDX,%TRLIB,$X::10
:PU,$TRLIB
:ST,$X,$TRLIB::10:5:-1
:PU,$X
:RU,P,&TROFF,,%TROFF::10
:RU,P,&TS,,%TS::10
:RU,P,&UP,,%UP::10
:RU,P,&US,,%US::10
:RU,P,&WZ,,%WZ::10
:RU,P,&X,,%X::10
:SV,0
```

Figure 16. (cont.)

For convenience, the name of the PASCAL compiler was shortened from PASCL to P. Also, for all PASCAL compilations, the relocatable file name was explicitly stated because the version of the PASCAL compiler used to develop the software would not place the output on the same disc cartridge as the source file was on. If the default specification ",,-" was used, the relocatable file was placed on the first cartridge defined (LU 2) rather than on cartridge 10 where the source is located. The typical output to the operator's console from the Pascal compiler is shown in Figure 17. Notice that the output of the Pascal compiler is assembler code, which must be assembled by the program MACRO. Also remember, even though a particular module appears to be small, any INCLUDE modules are also part of the source code and must be compiled. The rather large INCLUDE module [RECR3 is part of almost every TRAMCON module and must be compiled for each module in which it is included.

```
Pascal :  0  errors in file &X
Pascal :  Macro scheduled
Macro :   No  errors total
```

Figure 17. Sample screen output from Pascal compiler.

For the two library modules, &TRLIB and &MPLIB, the information displayed by the compiler is slightly different, as shown in Figure 18.

```
1    0   : $PASCAL 'TRAMCON Library, Ver. DEV', SUBPROGRAM, HEAP 2, HEAPPARM
S OFF$
0  *** Warning:  This feature is HP-1000 Pascal

Pascal :  0  errors and 1 warnings in file &TRLIB
Pascal :  Macro scheduled
Macro :   No  errors total
```

Figure 18. Sample screen from Pascal compiler for library module &TRLIB.

The warning is not fatal and refers to the compiler SUBPROGRAM directive, which is an HP enhancement to the Pascal system. The use of Pascal and Macro directives is discussed in detail in Section 8.4 of this manual.

The two Library modules, &TRLIB and &MPLIB, are compiled and indexed using the following FMGR commands:

```
:RU,P,&MPLIB,,%MPLIB::10
:RU,LINDX,%MPLIB,$X::10
:PU,$MPLIB
:ST,$X,$MPLIB::10:5:-1
:PU,$X
:RU,P,&TRLIB,,%TRLIB::10
:RU,LINDX,%TRLIB,$X::10
:PU,$TRLIB
:ST,$X,$TRLIB::10:5:-1
:PU,$X
```

The indexed module is placed in a temporary file $X because the version of LINDX that was used to develop the software would not return the unused disc space when a size of "-1" was specified. Both the problem with PASCAL and the problem with LINDX occurred in the versions that were used for development and may have been corrected in subsequent releases. The library files are stored with -1 for the size parameter so that they will be created without extents and use only the disc space they actually require.

8.2.3 SEGMENTING Large Programs

Figure 19 is a list of TRAMCON programs that are too large to fit in a single 32 K partition and therefore, must be segmented and loaded using the multi-level loader MLLDR. The HP-1000 F-Series computer using the RTE6/VM operating system is a 16-bit minicomputer with no code and data separation. With single 16-bit word addressing, the maximum address is 32767. If any program is larger than 32 K, it must be segmented or divided into 32 K portions that can overlay each other. For efficient operation, when possible, TRAMCON On-Line programs were kept small enough so that segmentation was not necessary.

A segmented program could be designated MEMORY or DISC-resident. A memory-resident program is wholly contained in memory when executing, while a disc-resident program has only the main portion and the current segment(s) loaded into memory at any time during execution. The advantage to memory residency is that no disc I/O is required when swapping segments. Keeping disc I/O to a minimum was a prime directive for software design and development. Because memory is limited, most of the segmented programs must be disc-resident.

The programs that run most often, PLRP and MTRP, are not only memory resident but also lock themselves into memory once they begin execution. Two memory partitions were made that are just big enough to hold these two programs. So far, when TRAMCON boots up, the sequence of execution of programs causes the operating system to place these programs into the desired partitions. If these programs are changed, their required partition size (indicated by the loader when the program is loaded) should be checked and the special partitions adjusted accordingly, either by regenerating or by using the On-Line configuration at bootup (see Section 15).

Before segmenting any of the programs listed in Figure 19, all of the software modules must be gathered together into one disc file. The modules are gathered together and indexed for quick reference using the HP supplied program INDXR (refer to RTE6/VM Loader Reference Manual, p. 6-41). The names of the files containing the various relocatable modules to be grouped together for a particular program are presented to the program INDXR in a command file. Naming conventions for these INDXR directive files are:

(1) They are five characters long
(2) The first character is "#"
(3) The second and third letters are the first two letters of the program being indexed (e.g., for INIT the file name is #INDX)
(4) The last two letters are "DX"

A list of these command files is presented in Figure 20.

1. CF - Data file initialization program.
2. DT - Master-to-master data transfer program.
3. INIT - TRAMCON on_line software initialization program.
4. MTRP - TRAMCON monitored segment response processor.
5. PLRP - TRAMCON polled segment response processor.
6. SR - Master to master time synchronization program.

Figure 19. List of segmented programs.

<u>INDXR Command File for CF - #CFDX</u>

```
CR,@CF::10
IN,%CF
IN,$TRLIB
IN,$PLIB
IN,$PLDH2
EN
```

<u>INDXR Command File for DT - #DTDX</u>

```
CR,@DT::10
IN,%DT
IN,$TRLIB
IN,$PLIB
IN,$FMP6
IN,%DSNRV
EN
```

<u>INDXR Command File for INIT - #INDX</u>

```
CR,@INIT::10
IN,%INIT
IN,$TRLIB
IN,$PLDH2
IN,$PLIB
IN,$FMP6
EN
```

<u>INDXR Command File for MTRP - #MTDX</u>

```
CR,@MTRP::10
IN,%MTRP
IN,$MPLIB
IN,$TRLIB
IN,$PLIB
IN,$PLDH2
EN
```

<u>INDXR Command File for PLRP - #PLDX</u>

```
CR,@PLRP::10
IN,%PLRP
IN,$MPLIB
IN,$TRLIB
IN,$PLIB
IN,$PLDH2
EN
```

<u>INDXR Command File for SR - #SRDX</u>

```
CR,@SR::10
IN,%SR
IN,%DSNRV
IN,%JULIN
IN,%GTTIM
IN,%CRSET
IN,$TRLIB
IN,$PLIB
IN,$FMP6
EN
```

Figure 20. INDXR command files.

The output of the program INDXR is an indexed collection of all the relocatable modules that are referenced by the specific program being indexed. The file naming convention used for these indexed files is

(1) The first letter is "@"
(2) The next one to five letters are the first to fifth letters of the executable program name (e.g., for INIT, the file name is @INIT).

File names can be seen in Figure 20 as the first directive for each program being indexed. These files should NOT be in existence when INDXR is run. INDXR will give a WARNING if the file specified in the CR directive already exists. It will probably be all right to overwrite this already existing file, but it is safer to purge the file before running INDXR.

In each file shown in Figure 20, the remaining directives are (1) an INCLUDE directive, that instructs the indexer program INDXR to include a particular RELOCATABLE module (%); or (2) an entire Library of modules ($) in the indexed file being created. Libraries $TRLIB and $MPLIB contain several user-created TRAMCON utilities that are described in Section 8.2.4. Library $PLIB is the main PASCAL library. Library $PLDH2 contains the routines to handle the type 2 HEAP (EMA). Library $FMP6 contains a few special routines used by programs DT, INIT, and SR.

NOTE

$SHSLB and %PRERS are not included in the segmenting and loading of these segmented programs. These modules are included in the linking of all the other TRAMCON On-Line software but would be most useful with these large programs. The library $SHSLB contains the short versions of the HEAP management routines and module %PRERS contains the short version of the Pascal run-time error reporter. Including either of these modules would reduce the size of the executable program and hopefully the number of segments. Refer to the Pascal/1000 Reference Manual, pp. 8-51, for a discussion of space savings. It is recommended that these routines be included with the segmented programs. To make the segmenter use the version of a specific routine from these modules instead of the Pascal library $PLIB, the directives "IN,$SHSLB" and "IN,%PRERS" would have to be added to the files in Figure 20 just before the directive "IN,$PLIB".

WARNING: Do NOT run INDXR with an incorrect command file name For example, the author has mistakenly run INDXR using the MLLDR command file #INIT instead of the INDXR command #INDX for the program INIT. If an incorrect command file name is used the error will be discovered immediately, because the indexer will display runaway error messages and the user will have to terminate the INDXR program. This not only confuses the program INDXR, it also corrupts one or more of the libraries such as $PLIB or $TRLIB, depending on which library was being indexed when INDXR was terminated.

Before the programs listed in Figure 19 can be loaded, they must be segmented. That is, they must be organized into executable pieces, none of which is larger than 64000 bytes, and all of which will execute by overlaying each other. The segmentation can be done by hand or by using a segmentation utility program, called SGMTR, supplied by HP. The segmentation job done by SGMTR proved adequate in all cases, so no hand segmentation has been done. For a complete discussion of the program SGMTR, refer to the RTE6/VM Loader Reference Manual, Chapter 6. Figure 21 shows the procedure for indexing and segmenting all the programs listed in Figure 19.

```
:***********************************
:** Index and Segment Program CF   **
:**
:PU,@CF::10
:RU,INDXR,#CFDX::10
:RU,SGMTR,@CF::10,#XX::10,29,CF,D
:RU,EDIT,#XX::10,TR,#E::10/
:PU,#XX::10
:PU,#CF::10
:RN,#Z::10,#CF
:***********************************
:** Index and Segment Program DT   **
:**
:PU,@DT::10
:RU,INDXR,#DTDX::10
:RU,SGMTR,@DT::10,#XX::10,17,DT,D
:RU,EDIT,#XX::10,TR,#ESRDT::10/
:PU,#XX::10
:PU,#DT::10
:RN,#Z::10,#DT
:***********************************
:** Index and Segment Program INIT **
:**
:PU,@INIT::10
:RU,INDXR,#INDX::10
:RU,SGMTR,@INIT::10,#XX::10,29,INIT,D
:RU,EDIT,#XX::10,TR,#E::10/
:PU,#XX::10
:PU,#INIT::10
:RN,#Z::10,#INIT
:***********************************
:** Index and Segment Program MTRP **
:**
:PU,@MTRP::10
:RU,INDXR,#MTDX::10
:RU,SGMTR,@MTRP::10,#XX::10,28,MTRP,M
:RU,EDIT,#XX::10,TR,#E::10/
:PU,#XX::10
```

Figure 21. Indexing and segmentation procedure.

```
:PU,#MTRP::10
:RN,#Z::10,#MTRP
:*************************************
:** Index and Segment Program PLRP **
:**
:PU,@PLRP::10
:RU,INDXR,#PLDX::10
:RU,TR,@PLRP::10,#XX::10,28,PLRP,M
:RU,EDIT,#XX::10,TR,#E::10/
:PU,#XX::10
:PU,#PLRP::10
:RN,#Z::10,#PLRP::10
:*************************************
:** Index and Segment Program SR   **
:**
:PU,@SR::10
:RU,INDXR,#SRDX::10
:RU,SGMTR,@SR::10,#XX::10,17,SR,D
:RU,EDIT,#XX::10,TR,#ESRDT::10/
:PU,#XX::10
:PU,#SR::10
:RN,#Z::10,#SR
:TR
```

Figure 21. (cont.)

After creating the indexed file of relocatable modules using the INDXR program, the segmenter program SGMTR is run to create the directives file to be used by the loader program MLLDR to load each of the six segmented programs. The various executions of the segmenter, as shown in Figure 21, require five run-string parameters.

The first parameter is the file name of the indexed relocatable module file previously created by program INDXR.

The second parameter is the file name of the output from the segmenter (#XX) that will contain directives to the loader.

The third parameter specifies the maximum segmentation path length in memory pages. This parameter is set to the relatively low value of 17 for programs DT and SR because they both use the DS software, which consumes a lot of memory and is not included in the segmentation process. By setting the path length to 17, room is allowed for the DS routines at load time. The other values are set to 28 or 29 out of a possible 30 to allow for some system overhead due to use of HEAP 2 (EMA - see RTE6/VM Loader Reference Manual, p. 6-3, paragraph 1).

The fourth parameter is the name of the executable program being segmented. The fifth parameter specifies whether each segment is to be memory or disc resident. Refer to the RTE6/VM Loader Reference Manual, p. 6-1, for further discussion of these run-string parameters.

To speed the loading process a bit and to reduce the size of the loader directives file, the segmentation output is placed into a temporary file called "#XX". This file is then edited with the source file editor, EDIT. The editor is told to look for instructions in the disc file whose name begins with "#E". These editor command files are text files that are interpreted by the editor as instructions on how to edit file #XX. Editor command files are listed in Figure 22.

The interpretation of the editor command files listed in Figure 22 is (1) Both files are text strings representing commands to the program EDIT, and (2) Each editor command is separated from the following command by either a "CR","LF" combination or the character "|".

The file #E begins with the editor string-search command "f" and instructs the editor to search for the string "TOTAL PROGRAM SIZE". A few of the comments, specifically the program size and the number of nodes (segments), are left in the segmenter output file for future reference. The information is contained on two lines in this file. When the first line (containing the string "TOTAL PROGRAM SIZE") is found, the next line is deleted with the "k" command and the two lines of data to be kept are joined into one line with the "j" command. This new line is temporarily marked with a first character of "=" so that it will not be deleted by the following commands. Also, any spaces on this line are temporarily changed to "_" with the "g/ /_/" command so that they will not be deleted by the following commands. Next, the loader "SH" directive is inserted after line 1, and a loader "LI" directive is inserted after line 3. The second line of file #E causes the editor to use regular expressions to delete all unwanted comment lines and spaces from the file being edited (refer to the RTE6/VM Loader Reference Manual, p. 6-18). Finally, the first character in the comment line that was protected is changed back to the "*" character and a new, edited version of the segmenter output is created with the file name #Z::10. For programs DT and SR, some additional editing must be done before that which was just described is done.

<u>File #E</u>

```
f/TOTAL PROGRAM SIZE/|/|k|-1|j|p=|g/ /_/|1|j| SH,SHAR1|3| LI,$PLDH2
sere on|3$x/ //q|3$d/^[A-Z,=]/aq|1|sewcl,1|f/=/|sewc|p*|g/_/ /|ec#Z::10
```

<u>File #ESRDT</u>

```
3| OP,BG| OP,SS|1|TR,#E/
```

Figure 22. EDIT command files for editing SGMTR output.

The additional editing is directed by the EDIT command file #ESRDT listed in Figure 22. The programs DT and SR use the DS software and, therefore, must be loaded as BackGround (BG) and have access to the Subsystem Global Area (SSGA). File #ESRDT places these two directives into the loader directive file, then transfers to file #E for the rest of the editing directives. The "/" character, which appears at the end of the only line in file #ESRDT and at the end of each "**RU,EDIT**" command shown in Figure 21, instructs the editor to transfer immediately to the command file specified without asking if it is all right to do so.

Since the editor created a new file, #Z, the old segmenter output file, #XX, can be purged with the FMGR command "**:PU,#XX::10**". Finally, the old MLLDR directive file is purged for the specific program and the newly created file, #Z, is renamed for the specific program, according to the naming convention mentioned above. For example, for program SR, the purge and rename instructions are "**:PU,#SR::10**" and "**:RN,#Z::10,#SR**". The indexed file ("**@SR**") and the loader directive file ("**#SR**") are now ready for the loading process discussed in Section 8.2.5.

8.2.4 Library Creation and Maintenance

A library is a collection of SUBROUTINES and FUNCTIONS that are referenced by more than one program module. These libraries can be compiled as separate modules and stored as relocatable code that is ready to be incorporated into an executable module. For Pascal library routines, to prevent the Pascal compiler from trying to interpret the library module as a program module, the PASCAL directive line must include the SUBPROGRAM directive.

The TRAMCON On-Line software package includes two libraries of routines. In keeping with HP file-naming conventions, any library module names begin with character "$". The two TRAMCON libraries are called $TRLIB and $MPLIB and are referred to in the literature as the TRAMCON library and the Monitor/Poller Library respectively. Library $TRLIB contains a collection of general-purpose routines used by most of the TRAMCON On-Line programs. Library $MPLIB contains routines used mainly by the response-processing programs PLRP and MTRP. Separating the routines into the two libraries speeds up the linking and loading processes because all routines do NOT have to be included in the search for externals each time a program is linked or loaded. That is, the library $MPLIB must be included when loading programs MTRP, PLRP or SW.

The routines described in this section were written in Pascal. Each description starts with the Pascal PROCEDURE or FUNCTION header showing the routine name and the formal parameter definitions. This header line is followed by a list of TRAMCON programs that reference this routine. In order to reference these routines, the program must include the formal procedure definition exactly as shown in these descriptions, followed by the directive "**EXTERNAL;**". For example, the procedure "**capitalize**" is represented in a program module by the statement "**capitalize (VAR ch: CHAR);EXTERNAL;**".

Parameters are passed to subroutines either by "**value**" or by "**reference**" (see HP Pascal Reference Manual, p. 4-31). Call-by-reference parameters are designated in the formal declaration by preceding the formal identifier with the reserved word "**VAR**". All other parameters are call-by-value. As explained in the manuals, changing a call-by-reference parameter will change the actual parameter in the calling routine, whereas changing a call-by-value parameter will change a local value, leaving the calling routine unaffected.

8.2.4.1 $TRLIB Routine Descriptions

Routines included in this library are referenced by more than one TRAMCON On-Line program. This library is included in the linking process for all TRAMCON programs.

PROCEDURE set_data_frame (segmentlu:INT;remotetype: remote_types);

Ref by POLL, MTRP, and CMMD (for monitored segments)

The "**set_data_frame**" procedure is used to program the BACI interface for the polling channels. The values set are Parity, Parity Sense, Data Bits, and number of STOP bits. The BACI interface must be initialized before each I/O request because these values are stored in RAM on the interface and power may have been lost since the last setting. These values were originally set with hard-coded commands before each POLLING or LISTEN request was issued because only one remote unit type was in use. This more general procedure was added to make the software flexible and able to accommodate other types of remote units. This routine sets the I/O interface specified by the "**segmentlu**" to the settings desired for the given "**remotetype**".

Program POLL calls this routine before each polling message is sent.

Programs MTRP and CMMD call this routine before issuing a LISTEN request on a given polling line. In this case, MTRP and CMMD can only guess at the "**remotetype**" by passing the type of the remote unit from which they logically expect to hear a response next. This is all fairly moot since only one remote unit type is being used. If the system is upgraded to a different remote unit in the future or upgraded to support remote units for other systems, such as the AT&T ACORN system or the Rockwell/Collins FACS system, then the software support is already in place.

PROCEDURE set_cat_vars;

Ref by process_response, update_cn, and update_al in $MPLIB, and LS

This routine can be used by any program to set the link-end category maintenance variables "**equip**", "**oppo_site**", "**siteptr**", "**linksptr**", "**linkendptr**", and "**local_end**" that are defined in the INCLUDE module [TRVAR. Considerable execution time can be saved by setting these values once rather than de-referencing the full identifiers each time they are referenced. This routine can be used instead of routine "**get_category**" if the full function of "**get_category**" is not required.

FUNCTION get_entry_address (entry_name: five_chars): INT;

Ref by SETCR

The Function "**get_entry_address**" is used to search the system and user entry point tables for a match of the five-character name passed in "**entry_name**". This table, which is stored on disc, starts at the disc address stored in the system communication area at memory address 1009 (1761B). The number of system entry points contained in the table is found at memory location 1012 (1764B) and the number of user-available entry points is found at memory address 1010 (1762B).

If a match is made, the memory address of the entry point is returned as the value of the function. If NO match is made, the value of the function is -1.

PROCEDURE off_prog (pname: six_chars; cloned: INT);

Ref by clone_and_run

This routine programmatically issues the operating system command OF to terminate the program specified by "**pname**". If the program is a clone, the value of "**cloned**" will be the CRT ordinal or index into the array "**current_crt**". If cloned, the program name is modified with the terminal LU number, which is located in the HEAP, by using the value of "**cloned**" as the index into the array "**heap^.current_crt[cloned]**". The cloned program name is composed of the first two characters of the parameter "**pname**" concatenated with the alphanumeric value of the terminal LU. For example, the name AL25 refers to a cloned copy of program AL and is composed by taking the first two letters of the program name AL and attaching the ASCII representation of LU number 25.

FUNCTION capitalize (ch: CHAR): CHAR;

Ref by CMMD

The value of character "**ch**" is passed to this routine by NAME. If this value is between the ASCII characters "`" and "{" (i.e., if it is any character from "a" to "z") then the corresponding capital letter is returned to the calling program as the value of the function.

FUNCTION ElapsedTime (resettimer: BOOLEAN): INTEGER;

Ref by poll_remote

The routine "**ElapsedTime**" is used for timing of various functions of the TRAMCON On-Line software. Timing values are kept separately for each TRAMCON segment. If this routine is called to start the timer ("resettimer" is true) for a segment (indicated by global "segord"), The local value of "**basetime**" for the given segment is set to the current time, otherwise the time elapsed since the last setting of basetime is returned as the value of the function.

FUNCTION TimeNow : INTEGER;

Ref by DT, INIT

This routine reads the system clock, converts the time/date into a two-word integer value representing the seconds since 00:00 1 January 1970 and returns the time/date in this form to the caller as the value of the function. Program DT calls this routine to time its data transfer operations.

PROCEDURE Day_Time (tm: INTEGER; VAR tm_str: twenty8_chars);

Ref by MA, SS

The routine "**Day_Time**" is passed a time/date value in the two-word integer format representing the number of seconds since 00:00 1 January 1970. The time/date is unpacked into the Julian day, year, month, day, hour, minutes, and seconds, is converted to ASCII, and returned to the caller in the string "**tm_str**". This is the time/date string displayed at the bottom of the MA and SS displays.

PROCEDURE poll_remote (caller, seg, rem: INT; cmd: CHAR);

Ref by PLRP, CMMD, PM, SW

Routine "**poll_remote**" formulates a FULL or NORMAL Poll request and sends it to the program POLL using a CLASS I/O call with the LU set to zero. The caller supplies only the segment ordinal, the remote ordinal, the "**cmd**" (FULL or NORMAL) and the callers ID in "**caller**". If polling transmission is being timed, the timer is started with a call to "**ElapsedTime**".

FUNCTION get_site_status (ss_dsp: BOOLEAN): two_chars;

Ref by CMMD, MA, PM, SS, update_displays in $MPLIB

This function returns a two-character value representing the most severe status indication for the site represented by the global values "**segord**" and "**remoteord**" as the value of the function. These indicators are displayed on the MA and SS displays just to the left of the site identifier. Programs MA and SS call this routine to refresh the entire display as it is being painted on the screen. Programs MTRP and PLRP call indirectly through routine "**update_displays**" to refresh the status indicator for one site. Programs CMMD and PM call to update the status indicators for all the remotes on existing MA and SS displays. The parameter "**ss_dsp**" is passed to this routine because the status indicator has slightly different values for the MA and SS displays. The site status returned as the value of the function can have any of the following values that are listed from most to least severe in Figure 23.

If a caller wants the site status returned for the SS display ("ss_dsp" is true), any status value greater than 4 is reported back to the caller as two BLANK characters. Otherwise, the caller wants the site status for the MA display that is returned according to the list in Figure 23.

0. NP - Remote unit NOT being polled.
1. NA - NO Answer from any of the PHYSICAL units constituting the LOGICAL remote unit.
2. na - NO Answer from at least one but NOT all of the PHYSICAL units constituting the LOGICAL remote unit.
3. PE - The BACI Interface detected a parity error in at least one byte of the response.
4. BR - Bad Response indicating that the remote unit ID in the response was unrecognized or the response length was invalid.
5. ME - At least one Major Equipment alarm was detected.
6. MS - At least one Major Site alarm was detected.
7. PA - At least one analog or digital parameter crossed a RED threshold.
8. mE - At least one minor Equipment alarm was detected.
9. mS - At least one minor Site alarm was detected.
10. pA - At least one analog or digital parameter crossed an Amber threshold.
11. ok - None of the above.

Figure 23. List of site status indicators for MA, amd SS displays.

PROCEDURE down_crt (devicedown: BOOLEAN);

Ref by CMMD, LO, crt_status_check

This routine is called when a program has failed to communicate with the terminal device indicated by global value "**crtord**". Certain terminal status indicators in the shared data area are set to prevent any software from attempting to communicate with this terminal until it is repaired. To further prevent any TRAMCON software from doing I/O to this device, the LU number used by the software to communicate is made to point to the bit bucket by issuing the operating system command "**LU,nn,0**", where nn is the terminal LU number. The program UP is informed through a CLASS I/O call that the terminal is down. Refer to Section 9.4 of this manual for further detail on the handling of downed terminals.

PROCEDURE crt_status_check (crtord: INT);

Ref by keypress, LO

The value of "**crtord**" is passed to this routine by name. Any time physical output to a CRT is completed, this routine is called to determine if the CRT is still operational by issuing a short write statement to the CRT pointed at by the value of "**crtord**". Originally this routine was widely used, but has been reduced to references in the few programs listed above for two reasons. First, most output to a CRT is now terminated with a call to the central routine "**key press,**" which has the call to built in. Secondly, the buffer size for the CRT text file "**outunit**" was increased to approximately one screenful, requiring fewer physical output statements. That is, several LOGICAL output statements are now issued by most programs before a "**writeln**" or "**prompt**" statement is issued. With the larger buffer, the operating system can buffer more information before actually sending it to the terminal device. The status of the device has to be checked only when data are actually sent to it.

PROCEDURE ring_audible (broadcast: BOOLEAN);

Ref by update_displays in $MPLIB

This routine rings the proper audible alarms located in the TRAMCON master cabinet. The alarms activated are determined by the values in the HEAP variable "**ss_alarms**" unless the parameter "**broadcast**" is true. IF "**broadcast**" is true, all the audible alarms are turned on to signal a catastrophic failure of the communication system.

FUNCTION read_dict (p:dictionary_ptr;VAR w:dictionary_word):INT;

Ref by most TRAMCON programs

This is one of the most frequently referenced routines. The value of VAR parameter "**w**" is set to the word from the data base dictionary, which is pointed to by parameter "**p**". Parameter "**p**" is actually an index into the large character array called "**heap^.dictionary**", that was placed in the HEAP at bootup. The length (the number of characters) of the dictionary word is returned as the value of the function. Notice that dictionary words are terminated within the dictionary by the ASCII character DELETE.

PROCEDURE disable_keyboard ;

Ref by CMMD, KYBRD, update_displays in $MPLIB

This procedure is called when a program module wants to have key presses ignored locally by the terminal and usually precedes output to the terminal. A terminal firmware flag that disables the keyboard is set by programmatically sending the Escape sequence "**Esc c**".

PROCEDURE save_cursor ;

Ref by MSG, update_displays in $MPLIB

This procedure was intended to allow a program module to interrupt another program's terminal input operation temporarily to display urgent messages. Once the message was displayed, the cursor would be returned to the position it had been in before the interruption. That cursor position was saved in the HEAP variable "**heap^.current_crt[crtord].last_cursor**" by calling this routine with "**crtord**" pointed to the appropriate terminal device. The cursor position is requested from the device by issuing the Escape sequence "**Esc a**" and reading the terminal's response into "**last_cursor**". An inconvenience with this process is that any input up to the point of interruption, is lost, but no indication of this loss is given to the operator. One possible correction would be to store the starting cursor position for each terminal input request and, after interruption, set the cursor back to the starting point and erase any previous echoes of this input.

PROCEDURE keypress (func: INT);

Ref by most TRAMCON programs

Procedure "**keypress**" is the most important procedure in this library and the most widely used of all the TRAMCON library routines. This routine is the heart of the terminal I/O system. Refer to Section 9 of this manual for details on the terminal I/O system design and how the routine "**keypress**" fits in. This procedure issues keyboard input requests using the CLASS I/O system so that the input information can be rigidly controlled and routed. The various options concerning the actual request are specified by the caller in the single parameter "**func**", which is passed to "**keypress**" when it is called. No information is returned directly to the caller. Instead, the keyboard input requests are attached to the appropriate CLASS number for the terminal specified by the value of the global variable "**crtord**".

Generally, the length (in characters) of the input desired is represented by the absolute value of parameter "**func**". If the request is for FUNCTION keys then the value of "**func**" will be the number of FUNCTION keys defined (from 1 to 8) added to 2000.

If it is a FUNCTION key request and the labels are displayed in big letters on the graphics display, then the value is the number of FUNCTION keys defined plus 2010. If it is a FUNCTION key request, procedure "**keypress**" will turn on the FUNCTION key menu, set the buffer length to one character (-1), set "**echo**" to false, and only allow the ASCII characters '1' to '8' as input. All the parameters defining a new input request are stored in the HEAP for the given terminal so that an interrupted or cancelled input request can be restored. A value of **-1 (nill)** for "**func**" is the most common call to "**keypress**". A program calls "**keypress**" with "**func**" set to nill when that program wants to put the terminal back into its IDLE state, the state where it waits for any key to be pressed. When a key is pressed, the program KYBRD is notified.

PROCEDURE check_more (short_pg: BOOLEAN);

Ref by AL, LS, ME, PA, PC

This procedure is called by any display program which produces information that is potentially more than one screenful. For instance, the AL display could be an arbitrary amount of lines long, depending on the status of the site being displayed. As each line of data is LOGICALLY displayed with "**write**" statements, this routine is called. The count of lines displayed is compared to the maximum number of lines allowed for this particular display, which is stored in the HEAP variable "**max_dsp_ln**". If the screen is not full, a LOGICAL line is completed by adding the characters LF and CR to the output buffer. If the screen is full, the LOGICAL output stored in the global I/O buffer "**outunit**" is PHYSICALLY sent to the terminal device by calling routine "**keypress**" which, in turn, issues the PHYSICAL output instruction "**prompt**". This call to "**keypress**" is a FUNCTION key request. That is, the terminal will be set up by routine "**keypress**" to prompt the operator for a FUNCTION keypress. A FUNCTION key request is indicated by adding 2000 to the parameter in the "**keypress**" call. Each program that uses this routine must set the global variable "**nbr_defined**" to a number between 1 and 8 to indicate how many of the possible eight FUNCTION keys are defined for use in this program. Refer to Section 9 for details on the terminal I/O processing. The value of "**nbr_defined**" is also added to the value of the parameter passed to "**keypress**".

Several bookkeeping values for the current display are stored in the HEAP variable "**heap^.current_crt[crtord]**. These values include "**pgs_remaining**", "**nbr_lines**", "**cur_page**", "**line_nbr**", "**prev_pages**", and "**lines[cur_page]**". Refer to Section 11.1 for details on the meaning of these values.

PROCEDURE wait_for_big_softkey (labels, ln2:soft_key_labels_type;
nbr_defined:INT;
paint_sc,clear_sc:BOOLEAN);

Ref by CO, DT, PM, SC, SR

This routine is called when a program wants to prompt the operator for a FUNCTION keypress and display the FUNCTION key menu in large text on the graphics display. The labels for each FUNCTION key are passed by the caller in the parameters "**labels**" and "**ln2**". Each key label can be two lines long with the first line contained in "**labels**" and the second line in "**ln2**". The menu is displayed in graphics TEXT mode with character size 3. Therefore, the label line size is limited to 26 characters for a 2397A terminal. Data on the alphanumeric display is left intact and turned off, so that the information can be redisplayed by sending a simple escape sequence later. Sometimes the graphics display already contains the FUNCTION key labels desired when the call is made to "**wait_for_big_softkey**", in which case the value of parameter "**paint_screen**" is false. Other times a few header lines are already on the graphics display, but the FUNCTION key menu is not. In this case, "**clear_screen**" is false, "**paint_screen**" is true and the FUNCTION key menu is displayed starting from the current cursor position instead of from "**whole_y - 60**". The actual keyboard input request is made by calling the routine "**keypress**" and signaling that this is a FUNCTION key request by adding 2010 to the parameter passed to "**keypress**". The parameter sent to "**keypress**" is negative to tell "**keypress**" NOT to release the terminal resource number, thus preventing any other programs from attempting I/O on this terminal. This routine processes the f1 key directly.

PROCEDURE jtime (VAR time_alfa: time_str);

Ref by CMMD

This routine reads the system time/date clock, converts the current time/date into the format ddd/hh:mm:ss and passes this string back to the caller in parameter "**time_alfa**". This routine was originally called by several modules throughout the TRAMCON software, but is now referenced by the single module

CMMD. The time/date string produced by this routine is displayed in the upper left-hand corner of most displays. Since this time/date stamp is common to most displays, the program CMMD displays this value just before it schedules the appropriate display program.

PROCEDURE time_date (graphx: INT);

Ref by HR, MA, SS

This routine reads the system time/date clock, converts it into an ASCII string indicating the day-of-the-week, day-of-the-month, month, and year and displays it on line 23 of the alphanumeric display or the text line indicated by the parameter "**graphx**" on the graphics display. The two displays currently showing this time/date string are the SS display (alphanumeric) and the MA display (graphic). Program HR calls this routine to refresh any existing SS or MA displays at midnight with the new day-of-the-week, day-of-the-month, and possibly, new month and year.

PROCEDURE display_current_msg ;

Ref by CMMD, MA, PR, SS

This routine is called to display any message that might be queued up for the terminal indicated by the global variable "**crtord**". Random messages to be displayed on a given terminal are stored, along with some message accounting values, in the HEAP.

PROCEDURE run_prog (f:INT; nam:six_chars; p1,p2,p3,p4,p5:INT);

Ref by clone_and_run

This procedure performs the following four steps (FMGR functions) necessary to programmatically run a type 6 program.

1. **OPEN** the type 6 disc file whose name is specified in parameter "nam".
2. **RESTORE PROGRAM (RP)**, which allocates an ID Segment to this program and places the executable portion of the program into the operating system scratch disc area.
3. **CLOSE** the type 6 disc file.
4. **SCHEDULE** the program for execution.

**PROCEDURE clone_and_run (nam:six_chars; p1,p2,p3,p4,p5: INT;
assign_to_partition: INT);**

Ref by CMMD, UP

This procedure programmatically executes the program whose name is specified in parameter "**nam**". The program is assumed to be a type 6 program. That is, the program specified may not be currently loaded and ready to execute. It is also assumed that the program is to be run as a CLONE of the actual program. The name of the cloned program executed is formed from the first two characters of the parameter "**nam**" concatenated with the alphanumeric representation of the LU for the terminal from which the request for program execution came. The terminal is specified by the value of the parameter "**crt_ord**". The HEAP record for the given terminal is checked to see if the desired program is currently loaded and ready to run ("old_dsp" matches "current_display"). If they are the same, the program is already loaded, and this procedure merely has to execute it by calling the routine "**schedule**", which is actually system procedure EXEC. If this is a new program for the given terminal, the old program whose name is in "**old_dsp**" is removed from both its ID segment and the system scratch track area by calling routine "**off_prog**". The value of "**crt_ord**" is passed to "**off_prog**" so that it will turn off the cloned program rather than a program with the raw name in "**old_dsp**".

To prepare a program for scheduling, the type 6 file must be opened, the program must be restored by issuing the FMGR RP command using routine IDRPL, and the type 6 file must be closed. At one time it was thought that perhaps it might be desirable to assign some programs to certain partitions. That function was not implemented for any TRAMCON programs, but the code remains for possible future use. Under RTE6/VM, a program can be passed five one-word values when the program is scheduled. These five values are accepted from the caller as parameters p1 through p5 and are simply passed on to the program scheduler in the call to "**schedule**". The schedule call is a 24, which is queued with NO wait (refer to HP Programer's Reference Manual p. 2-58).

PROCEDURE allocate_EMA (stack_needed: INT; VAR id: byte;
init_crt: BOOLEAN);

Ref by most TRAMCON programs

This procedure is referenced by any TRAMCON program wanting to access the shared data area called the HEAP. Refer to Sections 11.1 and 11.2 for a detailed discussion of the HEAP and how it is accessed. The call to this routine should be one of the first executable statements in any program but must be preceded by a statement that assigns the CLASS number associated with the first word address (FWA) of the HEAP to the global ([TRVAR) variable "**parms[1]**". Since the HEAP is a type 2 storage area, the HEAP addresses are two-word addresses. Therefore, the FWA of the HEAP is not passed directly to each program as it is scheduled. Instead, a one-word CLASS number was allocated by program CMMD and the FWA of the HEAP was attached to this CLASS number by issuing a CLASS WRITE once at system bootup. This CLASS number is passed as the first run-string parameter (parms[1]) to each program as it is scheduled. The run-string parameters are recovered and placed into global array "**parms**" (in [TRVAR) by calling the routine "**get_parms**". This call is usually followed immediately by a call to routine "**allocate_EMA**". Typically, the first statements in any TRAMCON program scheduled by CMMD are as follows:

get_parms(parms); allocate_EMA(0, id, TRUE);

The global VAR "**heap**" (in [TRVAR) is assigned the FWA of the HEAP by issuing a CLASS GET ("get_heap_ptr") on the CLASS number in "**parms[1]**" with "**heap**" as the two-word input buffer. The global "**heap**" now points to the entire HEAP described in Section 11.1, which means that all the information in the HEAP is part of the record "**heap^**".

The global variables "**crtord**" and "**segord**" are set to the values found in "**parms**[4]" and "**parms**[5]" respectively. These values were passed to the program by CMMD as run-string parameters. If the program is a display program ("init_crt" is true) that will be sending information to the terminal screen or printer, then "**allocate_EMA**" opens the global file "**output**" and attaches it to the terminal LU associated with the global VAR "**crtord**". The global VARs "**crt_type**", "**colored**", "**print_it**" and "**segptr**" are set to values found in the HEAP record "**current_crt**", that were set by the program CMMD before scheduling this program. These global values are set so that the program can reference the much more accessible variables (SCALAR GLOBAL) rather than two-word addressable fields within records.

Only two programs, MTRP and PLRP, require stack space to use for the handling of the recursive routine "**evaluate_node**" (refer to $MPLIB Routine Descriptions to follow). Stack space is allocated in increments of 50 words and is requested by passing a nonzero value for the parameter "**stack_needed**". If "**stack_needed**" is greater than zero, its value is a factor in determining the number of words of stack space to be allocated to the caller.

PROCEDURE deallocate_EMA (id: byte);

Currently NOT referenced

NOTE

This routine was intended for programs that wanted to return stack space that was no longer needed. As mentioned above, only the two programs, MTRP and PLRP, currently request stack space. These programs never terminate and, therefore, never return their stack space.

FUNCTION printer_status (repeat_cnt: INT): INT;

Ref by CMMD, print_display

This procedure attempts to discover the status of the printer that may or may not be attached to the external device port of the terminal identified by the value of global variable "**crtord**". The method for getting this information varies slightly in detail between terminal types, but the general procedure is the same. The Escape sequence "**esc &p4^**" is sent to the terminal asking for the printer status. The terminal responds with the seven-byte sequence "**esc \p4xxx**", where xxx is the three-byte device status. Figure 24 shows the manual references for the device status request for each terminal type supported.

Terminal Model		**Manual**	**Page**
1.	HP-2647F	2647F Reference Manual, Part No. 02647-90037	10-6
2.	HP-2627A	2627A Reference Manual, Part No. 02627-90002	8-6
3.	HP-2397A	2397A Reference Manual, Part No. 02397-90002	9-21

Figure 24. DEVICE STATUS information in terminal reference manuals.

The routine will repeat the status check as many times as required by the caller if the returned indication is "**busy**". The general printer status is returned to the caller as the value of the function.

PROCEDURE init_printer (compressed: BOOLEAN);

Ref by AL, CN, CR, LS, ME, PA, PC, PF, SE, WZ

If the printer is functional, then this routine is called to prepare the printer for output. First, the message "**Printing, Please WAIT**" is displayed on the CRT that is the current definition of text file "**outunit**". Since all TRAMCON programs have only one text file defined, the same file must be used for the printer. Therefore, the file "**outunit**" is redefined to point to the printer with the "**rewrite**" statement. Once "**outunit**" is redefined, any output statements to it will go to the printer. The printer LU number is computed based on the value of "**crtord**" instead of being stored, thus, it is important for the printer LU numbers to be consecutive. They are currently assigned to be LU numbers 42 through 45 and correspond to the terminal LU numbers 25 through 28, respectively. That is, the printer, referred to as LU 42, must be connected to the terminal referred to as LU 25. To compute the printer LU number, 42 is added to the value of "crtord", which ranges from 0 to 3. The printers are defined as subunits of their respective terminal equipment entries since they communicate over the same physical I/O port as the terminal device to which they are connected. The 2631G printers connected to the 2647F terminals use subchannel 5. The 2932A printers use subchannel 4.

PROCEDURE print_done ;

Ref by AL, CN, CR, LS, ME, PA, PC, PF, SE, WZ

This procedure is called when a program has finished its print job so that the file "**outunit**" can be redefined to be the terminal display and keyboard. Before "**outunit**" is reassigned, a Form Feed is sent to the printer. If the file "**outunit**" was not redefined to refer to the display and keyboard, the TRAMCON software could no longer communicate with this terminal device. As explained in Section 9, the keyboard is set to the "**wakeup**" state with a call to routine "**keypress (0)**".

PROCEDURE print_display ;

Ref by AL, HI, PH, PR, US, WZ

This routine is called when the TRAMCON operator wishes to copy the contents of the terminal display to the printer attached to the external device port of that same terminal. This function is performed locally by the terminal device after an Escape sequence is sent by the program to trigger this action. This function was difficult to accomplish for the older 2647F terminal because there was no indication sent from the terminal to the program when the job was done. The newer terminals send a single character back to the program when the job is finished. The software issues a single character read request to the terminal device and is suspended until completion of that request. The value of "**graphics_mode**" in the HEAP determines whether the graphics or alphanumeric display is copied to the printer. Some displays have a few lines of heading locked, which will prevent the cursor from being set to the top of the display. If some lines are locked, the lock is temporarily released with the escape sequence "**esc m**", the cursor homed, the display printed, and the lock reinstated with the Escape sequence "**esc l**".

8.2.4.2 $MPLIB Routine Descriptions

This library is known as the Monitor-Poller Library. This library was created because most of its routines are shared by the two remote unit response handling programs MTRP and PLRP. Actually, there is very little else to each of those two programs outside of this library. Also, since these routines are not used by any other TRAMCON programs, they were grouped into a separate library from the main TRAMCON library $TRLIB. This way, the linking process does not have to search through these routines when loading most TRAMCON programs. One other program, SW, does reference a few routines in this library. The general functions of the routines in this library are to process remote unit responses and refresh displays.

PROCEDURE pm_Initialize ;

Ref by MTRP, PLRP

Initialization steps that are common to the two programs MTRP and PLRP are done by this procedure. Both programs call this routine when they begin execution. First, the calling program is locked into the memory partition in which they are currently executing. Since both are completely memory resident, locking them into memory ensures that they will never be swapped to disc. The next step is to get the address of the HEAP by calling "**allocate_EMA(400,id,FALSE)**". The 400 in that procedure call requests a stack space of 200 words. This stack space is used to handle the execution of the recursive procedure "**evaluate_node**".

FUNCTION reverse_bits (from_bits: data_char): CHAR;

Ref by unpack_response and print_response

This routine is part of the remote unit response processing set of routines that follow. This routine is needed to place the data bits in the response from a DATALOK10 remote unit into a true sequential order. That particular type of remote unit reports 6 bits of information per byte in the response string. Some DATALOK10 modules, (FEC) cards in particular, report the 6 data bits in reverse order and must be reversed by this routine so that all the response bits are sequential. The raw response byte is passed as the only parameter "**from_bits**" and the reversed byte is returned as the value of the function.

```
PROCEDURE transform_ordinal (remote_typ:remote_types;alarmord:INT;
                                        resptype: response_data_types;
                                        VAR catagory,two_state_ord: INT);
```

Ref by unpack_response

This is an extremely important routine because it contains the code that is unique for each remote unit type. Much of TRAMCON's versatility in handling various remote unit types is shown here. To be able to handle a new remote unit type, a relatively minor amount of code must be added to this routine. Currently, TRAMCON supports the DATALOK10 Model 1E and the DATALOK10 Model 1D. These two units are treated as different types of remote units because their responses have a different structure. TRAMCON can be made to handle other remote unit types, such as the Rockwell/Collins FACS remote or the AT&T ACORN remote, by adding code to this routine and a very few lines of code to the procedure "**unpack_response**" described below.

This routine transforms the data bits in the PHYSICAL remote unit response into their corresponding position in the generic data base equipment record. The bit position in the remote unit response is indicated by the value of parameter "**alarm_ord**". This "**alarm_ord**" is associated with a position in the equipment record by returning the two values "**category_ord**" and "**ord_in_cat**".

PROCEDURE unpack_response ;

Ref by MTRP, PLRP, SW

The purpose of this routine is to convert the raw responses received from any currently supported type of remote unit into a generic, unpacked format. The type of remote unit is meaningless beyond this routine. That is, all responses look the same to the response processing code after this routine is called. The raw response is in the global variable "**response**" and the unpacked response is placed into the global variable "**unpacked_response**" by this routine. The only code in this routine that is dependent on the remote unit type is the check for legal response length. If new remote units are added to the list supported by TRAMCON, a check for its legal response lengths must be added to this routine. For speed and convenience, the remote unit packs its information. This routine unpacks this data so that the response-processing code can get at the individual data bits more easily.

FUNCTION evaluate_node (tree: expression_tree;node: INT): BOOLEAN;

Ref by process_response

This routine is the only recursive routine used in the TRAMCON software. It is called to process the combination alarms that are defined by a LOGICAL expression tree in the equipment record.

PROCEDURE archive_it (noans: BOOLEAN);

Ref by update_displays

This routine is called to create an archive record for a given LOGICAL remote unit. To ensure consistency in the archive file, if an archive record is created for any one of the PHYSICAL remote units that make up a LOGICAL remote unit, this routine creates an archive record for all the PHYSICAL remote units that make up the given LOGICAL remote unit. The LOGICAL remote unit concept was added to the TRAMCON system late in its development, at which point, the archive file was organized by individual PHYSICAL remote units. To minimize changes, this structure was maintained even with addition of the multiple remote unit idea. Now, instead of an archived event for a given remote unit occupying one record, this same event occupies as many records as there are PHYSICAL remote units in the given LOGICAL remote unit. Refer to Section 6.4 of this manual for a discussion of PYSICAL and LOGICAL remote units. The NO ANSWER indication was expanded to indicate (by subtracting 5000 from the year of the main remote unit's archive record) that at least one, but not all, PHYSICAL remotes of a LOGICAL remote failed to answer. This indication is shown on the Alarm Archive display as "**na**". If none of the PHYSICAL remotes answered, 10000 is subtracted from the year of the main remote unit's archive record and this is indicated on the AR display as "**NA**".

Also important to note is the manual control of access to the archive file. This was implemented by hand, instead of opening the file exclusively, to avoid the costly overhead of constantly opening and closing the file. The exclusive use is managed by using the "**next_archive_record**" pointer, which excludes simultaneous use of only the section of the archive file that belongs to the given remote unit.

PROCEDURE parm_def (p_type: INT);

Ref by CC, PA and process_parm

For each UNIQUE analog or digital parameter monitored by the TRAMCON software, there is an IF-THEN-ELSE clause added to this routine. Given the type of the parameter in "**p_type**", this routine sets global variables (in [MPVAR) "**decimal_places**", **two_sided_th**", "**decreasing**", **and** "**calibrate**", which uniquely define a particular type of parameter. This does not mean that each time a new model of communication equipment is monitored lines of code must be added to this routine. Code needs to be added to this routine only if there is a parameter (e.g., RSL, Signal Quality) generated by this new equipment that does not match the settings of the four variables listed above for any of the currently monitored parameters. For example, when the FRC162 radio was added to the list of equipment that TRAMCON monitors, the characteristics of the RSL parameter for this radio were matched against the existing parameter types, namely types 1, 2, 3, and 60. The FRC162 RSL parameter has no decimal places, is not two sided, is not decreasing, and must be calibrated. This did not match the settings of any of the existing parameter types and, therefore, required that code be added to this routine to handle a new parameter type. These parameter types must be exactly coordinated with the data base Configurator so that the parameters are defined with the proper type in the data base. A list, like that in Figure 25, of the various types and their associated settings should be kept by the Configurator user so that reference can be made to it when introducing new communication equipment to be monitored by the TRAMCON system.

Notice that the new type, type 4, added to define the FRC162 RSL and BDM parameters, differs from the DRAMA RSL parameter type, type 1, in only one column, "**decreasing**". But one difference is all it takes to define a new parameter. In case any general distinctions need to be drawn between all analog and all digital parameters, any digital parameter type should be assigned a number greater than 59.

The value of "**decimal_places**" determines how many digits are displayed to the right of the decimal point for the given parameter. For ease of use (making calculations) and minimal memory storage requirements, all the parameter data are stored and used in one-word integer (INT) form. If a given parameter has two decimal places, the integer value is stored in centi-units. For example, DRAMA signal quality is parameter type 2 and, therefore, has two decimal places and its units are volts. So, a signal quality value of 456 represents a signal quality of 4.56 volts.

The "**decreasing**" value refers to the sense in which the parameter behaves in going from good (operating properly) to bad (malfunctioning). That is, does the raw parameter value increase or decrease when going from a good to a bad reading. This needs to be known by the parameter processing code so that it can make the correct relational tests (less than or greater than) to determine threshold crossings and calibrated values.

TYPE	decimal_places	decreasing	calibrate	two_sided_th	Examples
1	0	TRUE	TRUE	nill	DRAMA RSL
2	2	TRUE	FALSE	nill	DRAMA Sig Qual
3	2	TRUE	TRUE	5000	Site Battery
4	0	FALSE	TRUE	nill	FRC162 RSL, BDM
60	0	FALSE	FALSE	nill	Digital
61	0	FALSE	FALSE	nill	MD-918 #1 Err Rate
62	0	FALSE	FALSE	nill	MD-918 #2 Err Rate

Figure 25. List of parameter TYPES and their settings.

The "**calibrate**" characteristic determines whether the parameter can be used just as it is reported (uncalibrated) or must be converted to a different magnitude and/or units. For example, the site battery is a voltage value, and all analog parameters are reported by the remote units as voltages. But, the absolute value of the battery voltage is beyond the range of the DATALOK10 remote unit A/D module. Therefore, the site battery raw voltage value must be calibrated. In this simple case, the voltage is multiplied by 3. In other cases, the conversion is not linear, and the raw voltage value must be compared against a table of numbers called a calibration curve.

The last characteristic, "**two sided**", refers to the property of a parameter such that it is good (functional) at a NOMINAL value and degrades in performance both above and below that value. The only example of such a two-sided parameter so far is the site battery, which operates around a NOMINAL value of 50 volts. If a parameter is not two sided, the value of "**two_sided_th**" equals -1. Otherwise, the value of "**two_sided_th**" is equal to the NOMINAL value in centivolts (e.g., NOMINAL value = 50 volts, "two_sided_th" = 5000).

PROCEDURE print_parm (val: INT);

Ref by CC, PA, print_val

This procedure displays or prints the value of an analog or digital parameter contained in "**val**". The total field width for the value displayed or printed is six character positions, including any decimal point. The number of digits (ranging from 0 to 3) is indicated by the global VAR "**decimal_places**" (in [MPVAR) which must be set to the desired value before calling this routine. The number of decimal places for a given parameter is acquired by "**process_response**" before calling "**print_parm**" by calling the routine "**parm_def**", which is discussed above.

PROCEDURE process_response ;

Ref by MTRP, PLRP

Just as "**keypress**" is the most important routine in the keyboard input handling function, so this routine is the heart of the remote unit response processing function. Each time a response is received from a PHYSICAL remote unit, this routine is called to process the response.

Notice that this routine processes each PHYSICAL remote unit response. The concept of LOGICAL or MULTIPLE remote units was introduced with as little code change as possible. These response-processing routines were left as is rather than trying to rewrite them to process more than a single remote's response at a time. Refer to Section 6.4 for details on the processing of LOGICAL remote unit responses.

First, the routine "**unpack_response**" described above is called to transform the actual response as received from the remote unit (identified by the values of global VARs "**segord**" and "**remoteord**" in [TRVAR) into the generic response format defined by the TYPE "**unpacked_response_record**", which is defined in Section 11.1. The generic response is placed into global VAR "**unpacked_response**", which is declared in the INCLUDE module [MPVAR. The various sections in the response, two-states, analog and digital parameters, are matched against the previous response from the same remote unit, which is stored in the HEAP VAR "**rem_status_ptr^.cat_status**". The Configuration data contained in the HEAP records "**equip^**", "**segptr^**" **and** "**remptr^**" are used as templates to define the response being analyzed. For the given remote unit (remptr^) on the given segment (segptr^), these Configuration data describe which categories are defined and what type of equipment (equip^) is being monitored by each category.

If any difference is found between the previous response and the current response, it is flagged as a change-of-state, which will trigger the routine "**update_displays**" to call "**archive_it**" to create an archive record once the entire LOGICAL remote unit response has been analyzed.

Each analog and digital parameter value is processed by the local routine "**process_parm**" for every response received. Routine "**parm_def**" is called to set the global VARs "**two_sided**", "**decreasing**" and "**calibrate**". These global values are used by "**process_parm**" to determine the sense (less than or greater than) for the Logical comparisons that are made to determine the proper histogram BIN for the new value. The command "**calibrate**" is used to determine if the analog parameter value must be converted to a new value and new units other than volts.

PROCEDURE clear_chars (num_chars: INT);

Ref by update_cursor and update_displays

In several places throughout the TRAMCON software, text is written to the screen in graphics TEXT mode. Sometimes this text information overwrites existing text. To reduce the volume of data sent to the screen, a particular TEXT drawing mode, JAM4, was used that affects the entire character grid by turning off all bits in the grid except those that form the character. The result was the erasure of the old text and the displaying of the new. This JAM mode works fine for the first two models of terminals used by TRAMCON, but was rewritten for the HP-2397A. On the newer terminal, JAM4 mode now works exactly the same as JAM2 mode, overwriting but not erasing the old data. To remedy this situation for the MA display on a HP-2397A terminal, this routine was added to erase the old data by writing ASCII blank characters in JAM4 mode.

PROCEDURE update_cursor ;

Ref by MTRP, PLRP

This procedure is used by the response processing programs MTRP and PLRP to refresh the polling cursor on all the existing SS and MA displays. All terminals are examined for either of these displays by setting global variable "**crtord**" to zero and looping while "**crtord**" is less than "**max_crt**". Program PLRP, since it is polling the remote units, calls this routine when it is preparing to poll another remote unit. If a cursor already exists on a given display (heap^.current_crt[crtord].y <> 0), it is erased. Program PLRP then displays the new cursor next to the remote unit it is preparing to poll. Since it is only listening for responses, program MTRP does not know ahead of time which remote unit is being polled, if any. Therefore, program MTRP calls this routine just after it has received a response from any remote unit. Program MTRP writes the cursor next to the remote unit that just responded. In any case, the cursor for a given display cannot be updated if the terminal's resource number is locked to another process. For more information on the handling of the terminal I/O, refer to Section 9 of this manual.

PROCEDURE update_us ;

Ref by MTRP, PLRP

This routine is called to refresh existing statistics displays that were produced by the US command. Specifically, two sets of statistics, referred to as page2 and page3, are refreshed by this routine. The page2 display shows a breakdown of the kinds of remote unit responses registered by the software. The page3 display shows a breakout of remote unit response processing times. Refer to Section 12.2 for details on these data. As in all update or refresh routines, this routine must successfully lock the terminal resource number before it can refresh the display. This routine also checks all terminals for the US display, page2 or page3.

PROCEDURE update_ss (up_date: BOOLEAN);

Ref by SS, update_displays in $MPLIB

This routine refreshes the SS display on the terminal specified by the global variable "**crtord**". The routine is called to display the segment status information for the remote unit specified by the global variable "**remoteord**" and the segment specified by the global variable "**segord**". The display cursor is assumed to be on the appropriate line. The parameter "**up_date**" indicates whether this is a call to refresh an existing display (up_date = true) or paint a new display (up_date = FALSE). The program SS paints a new display and calls this routine after it has displayed the remote unit name and placed the display cursor in the proper column. If "**up_date**" is false, the display cursor must be positioned to column 30 by this routine before data can be refreshed. The data used for this display is stored in the HEAP variable "**heap^.segment_status[segord].remote_status[remoteord]^.ss_alarms**".

PROCEDURE update_cn ;

Ref by update_displays in $MPLIB

This routine is called to refresh an already existing CN (CouNted two-state) display for the remote unit specified by the global variable "**remoteord**" on the segment specified by the global variable "**segord**" on the terminal specified by the global variable "**crtord**". Since this display is potentially more than one screenful (page), the number of the page currently being displayed is kept in the HEAP variable "**heap^.current_crt[crtord].cur_page**".

This routine examines the static Configuration data for the given remote unit, which is kept in the HEAP, for any two-state value designated to be counted, starting with the SITE category two-states. Once a counted two-state is discovered, this routine decides whether this value is currently displayed or not. If this routine has passed by enough counted values to get to the current page, the value is refreshed on the screen. If not, the page line counter "**ln**" is incremented and, if necessary, the local page number, "**loc_pg**", is incremented until the local page equals the displayed page.

PROCEDURE update_pc ;

Ref by update_displays in $MPLIB

This routine is called by routine "**update_displays**" to refresh any **PC** display on any of the terminals on the given master. The **PC** display is a multi-page display and the current page and line information for a given terminal are kept in the HEAP record "**current_crt[crtord]**" in the fields "**first_line[]**", "**cur_page**", "**nbr_lines**" and "**line_nbr**".

All the trunks defined for the given segment that are displayed on the given CRT are examined to see if they begin and/or end at the site that just responded (siteptr). The first trunk displayed is indicated in "**first_line[cur_page-1]**" and the last trunk is indicated by the value "**nbr_lines**". If the responding remote unit (siteptr) matches the trunk start (site1) or the trunk end (site2), the value is updated on the screen.

PROCEDURE update_al ;

Ref by update_displays in $MPLIB

This routine is called by routine "**update_displays**" to refresh any AL display on any of the terminals on the given master. The AL display is a multi-page display and the data are displayed in CATEGORY order. But, the contents of each CATEGORY can change with each response. This makes it very difficult to refresh an existing display. Therefore, to refresh this display, the entire display is started over from the beginning.

PROCEDURE print_val (p_type,cat,rem: INT; val2,online:BOOLEAN);

Ref by update_pa in $MPLIB

This routine is called by routine "**update_pa**" each time a parameter value needs to be displayed on a given terminal. The value "**p_type**" is passed to the routine "**parm_def**" to set the values "**two_sided_th**", "**decreasing**", and "**decimal_places**". Routine "**print_parm**" is called to display the parameter value. Immediately following the value, the threshold crossing indicators are displayed.

PROCEDURE update_pa ;

Ref by update_displays in $MPLIB

This routine is called by routine "**update_displays**" to refresh any PA display on any of the terminals on the given master. The PA display is a multi-page display with each CATEGORY constituting a new page. The current display information is stored in the HEAP record "**current_crt[crtord]**". The current page (CATEGORY) is stored in field "**misc1**". If both ends of a given CATEGORY are monitored on the given segment, the information for the opposite end is stored in the fields "**misc2**" (remoteord2, identifying the opposite end remote unit) and "**misc3**" (category2, identifying what part of the opposite end remote unit is connected to this end).

PROCEDURE print_response (all_CRTs: BOOLEAN);

Ref by MTRP, PLRP

NOTE: THIS ROUTINE CONTAINS CODE SPECIFIC TO THE TYPE OF remote unit. ANY TIME A NEW remote unit TYPE IS SUPPORTED, THIS ROUTINE MUST BE MODIFIED ALONG WITH THE PREVIOUSLY MENTIONED ROUTINES "transform_ordinal" AND "unpack_response" IN THIS LIBRARY. ANY DEVICE OTHER THAN A DATALOK10 MODEL 1D OR A DATALOK10 MODEL 1E CONSTITUTES A NEW remote unit TYPE.

Unlike the procedures **"transform_ordinal" and "unpack_response"** mentioned above, which contain code dependent on remote unit type, this routine is NOT essential to the primary function of TRAMCON. This routine is mainly a convenience and a troubleshooting device for software development and enhancement. This routine displays a formatted raw response from a given remote unit specified by the global variable "**remptr**" on a given segment specified by the global variable "**segptr**". The graphics display was chosen for these data since more data could fit on one screen. This greater density in data is achieved at the cost of greater time to paint the display. It would be a relatively minor change to put the data on the alphanumeric display rather than the graphics display if that is desired in the future.

This routine may be requested to scan all terminals for the **DI** display (all_CRTs = true) or to display the response on a terminal specified by the global variable "**crtord**" (all_CRTs = false). If the request is for a specific terminal, the terminal device is already under the control of the caller. If the request is for all terminals, each terminal device must be secured for the exclusive use of this routine by LOCKING the resource number associated with this terminal. Only then can this routine be assured that no other process will disrupt the output of this routine. Refer to Section 9 of this manual for a detailed discussion of the terminal I/O management.

To speed things up a bit, a REFRESH mode was added whereby the static information on the screen is not refreshed each time. This routine determines whether to REFRESH or REPAINT the entire display by the value in the HEAP variable "**heap^.current_crt[crtord].current_display**". If the current display value is "**DI**", the screen should be REFRESHED, otherwise, the entire screen should be REPAINTED from scratch. Because of the graphics TEXT mode problem with the HP-2397A terminal, the entire screen is repainted each time by setting "**refresh**" to false and clearing the graphics display.

The data are grouped into types of data and displayed in the order they are received in the response. Data types are such things as "**two-state**" and A/D. The two-state information is displayed as a "1" or a "0". The A/D or digital data are displayed as real numbers representing the raw voltage readings. Hard-coded remote unit dependent information includes such things as (1) byte positions where each of these data types starts within the response, (2) the number of bytes that compose an A/D value and, (3) if encoded, what encoding method is used. For example, the DATALOK10 remote unit reports each A/D value in five bytes as BCD digits in the format shown in Figure 26.

In Figure 26 the actual voltage reading is encoded as three BCD digits represented above by "**U**" for the units digit, "**T**" for the tens digit, and "**H**" for the hundreds digit. A value greater than 999 is represented by setting the HD bit to a mark (M). The sign of the voltage value is indicated by the PO bit, which is a mark for positive and a space for negative. Any bit shown as a "**S**" or "**M**" above is a fixed space or mark, respectively.

BYTE	NAME	bit	1	2	3	4	5	6	7
1	Point char		S	M	M	M	M	M	M
2	Scale char		M	M	M	M	M	M	M
3	MSD char		H	H	H	H	HD	S	S
4	TENS char		T	T	T	T	PO	S	S
5	LSD char		U	U	U	U	OV	S	S

Figure 26. DATALOK10 A/D data format.

PROCEDURE get_answer (caller: INT);

Ref by MTRP, PLRP, SW

The routine "**get_answer**" is called by any program wanting to accept a RAW remote unit response. The calling program identifies which response it is interested in by specifying its unique CLASS number as the value of "**caller**". All remote unit response read statements are issued either by the program POLL for polled segments or by the program MTRP for monitored segments. In either case, the read is a CLASS READ attached to these same unique CLASS numbers. Routine "**get_answer**" issues a CLASS GET on the CLASS number "**caller**" to acquire any response attached to this CLASS number. Several global VARs are set to indicate the overall status of the response just received. These VARs are declared in the INCLUDE module [MPVAR. The VARs indicate whether any problems were encountered in the transmission of this response and include the following:

"**response_timedout**"	No response data were received within a preset time limit: 1 second for Polled Segments 1 minute for Monitored Segments.
"**parityerr**"	At least one byte in the response had a parity error as indicated by the BACI interface.
"**res_len_ok**"	Response length (in bytes) is within the range 2 to "max_chars_per_response".
"**illegal_interrupt**"	This is reported by the response channel driver, DVA76 or DVA77, when NO valid reason for entering the driver could be found.
"**cos**"	This is reported by the DATALOK10 remote unit in bit 5 of the second byte in the response to indicate if there is any change-of-state in this response.
"**bad_id**"	This value is true if the remote unit ID received in the first two bytes of the response does NOT match any remote unit ID in the Configuration data base on the given segment (specified by "segord").
"**responded**"	This value is a combination of the four previously determined values "res_len_ok", NOT "bad_id, NOT "parityerr", and NOT "response_timedout".

The global VARs "**segptr**", "**remptr**" and "**rem_status_ptr**", all declared in the INCLUDE module [TRVAR, are set as a convenience so that subsequent code does not have to perform the lengthy de-referencing of these values that are stored in the HEAP.

The routine "**read_clock**" is called to set the global VARs "**clk**" and "**yr**" (declared in [MPVAR) to the current time/date. This time/date value will be used by the response processing routines as the time stamp for any events that are new in this response. This is the time stamp found in the archive records in file (ARCH.

The HEAP VAR "**di_segrem**" is checked to see if the raw response just received is to be displayed. If so, the routine "**print_response**" is called to display the response just received on any display that wants to look at it.

PROCEDURE update_displays ;

Ref by MTRP, PLRP

Once a remote unit response is completely processed, all the terminals currently defined on the system are scanned for possible displays that can be refreshed. Figure 27 shows the displays that are refreshed by this routine. Each display listed has a separate routine that handles the updating of the particular display.

1. AL - Alarm/Status
2. CN - Counted two-states
3. MA - Segment MAP
4. PA - A/D and Digital Parameters
5. PC - Digroup Alarms
6. SS - Segment Status

Figure 27. Displays that are updated by "update_displays".

The refreshing of display US and the updating of the polling cursor are not done by this routine because those functions are not performed immediately after the response is processed. For example, the US display includes the timing of display refreshing, which is not complete until this routine executes and should not include itself in the timing. The polling cursor is refreshed before the response processing begins and this routine is executed after the response is processed.

The resource number "**crt_rn**" is used to properly manage the terminal I/O. This routine attempts to lock the "**crt_rn**" for each terminal for exclusive use. If the EXCLUSIVE LOCK is not successful, this routine does not attempt to refresh the terminal display.

Each terminal display, indicated by the HEAP VAR **"current_crt[crtord].current_display"**, is examined to see if it is a display that can be refreshed.

8.2.5 Linking and Loading

All TRAMCON programs are loaded and stored on the computer as type 6 programs as explained in Section 3.3. Most programs are small enough to fit in the 32 K address space. The program LINK is used to load these small programs. Program LINK produces a type 6 program by default and removes the program from active status (equivalent to the operating system "OF" command) after linking. The loading process was sped-up considerably with the LINK program. For the larger programs that must be segmented, the multilevel loader MLLDR is used. Program MLLDR is slow, but steps have been taken to speed-up the loading process also. Steps that have been discussed in this section, are listed in the RTE-6/VM Loader Reference Manual, p. 8-12. The first three steps discussed on page 8-12 have been taken.

Figure 28 shows the three LINK command files used to link most of the TRAMCON programs. The majority of the TRAMCON programs make use of the shared memory "**sharl**" and can be linked as the default type extended background (eb). These programs are linked using file "#". A few programs do not use the shared memory but must be linked as background (bg) because they reference items in Table Area II, which is not included in the address space for "eb" programs (see RTE-6/VM Loader Reference Manual, p. 2-14). These programs should be linked using LINK command file "##". One program, TS, uses command file "**###**" because it uses the shared memory and must be linked as "**bg**". In all cases, the shorter run-time error reporting routines in module "**%prers**" are used to reduce the size of all the programs (see Pascal/1000 Reference Manual, pp. 8-51). In all the above, the Library $TRLIB is assumed to be included in the system SNAP file (Refer to Section 10.3 of this manual).

Figure 29 shows the procedure for linking all nonsegmented TRAMCON programs. Before linking any program as in Figure 29, the program must be deactivated using the operating system "OF" command.

If a program, or any clone of that program, is active (occupies an ID segment), the linker will ABORT with an error indication similar to the following:

```
Program is active  AL::2:6
  Program name not usable
  Last module relocated: PAS.ERRORPRINTER
  Fatal Error 178 - Link terminated
```

<u>LINK Command file #</u>

```
sh,sharl
re,%prers
en
```

<u>LINK Command file ##</u>

```
op,bg
re,%prers
en
```

<u>LINK Command file ###</u>

```
op,bg
sh,sharl
re,%prers
re,%t2
en
```

Figure 28. LINK command files - #, ##, ###.

```
:SV,4
:RU,LINK,%AL,#
:RU,LINK,%ARPTR,#
:RU,LINK,%BROAD,#,%DSNRV
:RU,LINK,%CC,#
:RU,LINK,%CHECK,#
:RU,LINK,%CMMD,#,+RO,+SZ:30
:RU,LINK,%CN,#
:RU,LINK,%CO,#
:RU,LINK,%CR,#
:RU,LINK,%CURVU,#
:RU,LINK,%ED,#
:RU,LINK,%HI,#
:RU,LINK,%HR,#
:RU,LINK,%KYBRD,#
:RU,LINK,%LO,#
:RU,LINK,%LOF,##
:RU,LINK,%LON,##
:RU,LINK,%LS,#
:RU,LINK,%MA,#
:RU,LINK,%ME,#
:RU,LINK,%MEDX1,#
:RU,LINK,%MEIDX,#
:RU,LINK,%MS,#
:RU,LINK,%MSG,#
:RU,LINK,%PA,#
:RU,LINK,%PC,#
:RU,LINK,%PF,#
:RU,LINK,%PH,#
:RU,LINK,%PM,#
:RU,LINK,%POLL,#
:RU,LINK,%PR,#
:RU,LINK,%RMAST,#
:RU,LINK,%SC,#
:RU,LINK,%SE,#,+SZ:16
:RU,LINK,%SETCL,##
:RU,LINK,%SETCR,##
:RU,LINK,%SETDT,##
:RU,LINK,%SETVE,##
:RU,LINK,%SI,#
:RU,LINK,%SS,#
:RU,LINK,%SW,#
:RU,LINK,%TIMPA,#
:RU,LINK,%TIMSE,#,%CRSET,+BG
:RU,LINK,%TROFF,#
:RU,LINK,%TS,###
:RU,LINK,%UP,#
:RU,LINK,%US,#
:RU,LINK,%WZ,#
:RU,LINK,%X,#
:SV,0
```

Figure 29. Linking nonsegmented programs.

Program CMMD has the reorder command (+ro) in the LINK run-string because it is rather large and runs out of links if the modules are not reordered. Unless the programs listed in Figure 29 significantly increase in size, they will always appear to successfully link because the linker will use as many pages of memory as necessary. There have been a few times when a program that appears to link successfully, terminates with a fatal error such as "**EM82**" or "**MP**" when it is executed. By adding another memory page to the size determined by the linker, these fatal executions have been avoided. A best guess is that the linker will squeeze a program into the absolute minimum space necessary to the nearest memory page, disregarding any overhead required by the operating system. If the unused portion of the last page is very small, some code may get clobbered by the operating system as the program executes. A few of the programs have already encountered this problem and were fixed by specifying a program size (+sz) in the run-string when linking. As programs change, these explicitly-stated sizes may need adjustment, or may no longer be required.

Figure 30 shows the FMGR procedure file for loading the six segmented programs listed in Figure 19. Before each program can be loaded, it must be removed from active status using the FMGR "**OF**" command, which ensures that the program is removed from any ID segment it might occupy and is removed from the system scratch area on disc LU 2.

The multilevel loader, MLLDR, is used to load each of these segmented programs and is given loading directions in the file whose name appears in the MLLDR execution statement (e.g., "RU,MLLDR,#SR" says, "load program SR according to directions given in file #SR"). These loader directive files were discussed previously in this section and a sample of file #MTRP for loading program MTRP is shown in Figure 31.

Figure 31 shows the loader directive file #MTRP used by program MLLDR to load program MTRP. Notice the two comment lines that indicate how and when this file was created with the segmenter program SGMTR. Notice also how large the segmented type 6 program file is and how many segments (nodes) were created by the segmenter. This information is very useful when re-segmenting and reloading the particular program because of changes or enhancements. A prime directive used throughout the TRAMCON software development was to keep the number of segments to a minimum. This directive gives some assurance that the amount of segment swapping is kept to a minimum. Segment swapping is very important for the disc-resident programs and of minor concern for the two memory-resident programs MTRP and PLRP, since memory swapping is three orders of magnitude (1000 times) faster than disc swapping. But this directive is more important in keeping the overall size of the program to a minimum, which is very important for the memory-resident programs and of less importance for disc-resident programs.

```
:SV,4
:**********************
:** Load Program CF   *
:OF,CF
:RU,MLLDR,#CF
:PU,CF
:SP,CF
:PU,@CF
:**********************
:** Load Program DT   *
:OF,DT
:RU,MLLDR,#DT
:PU,DT
:SP,DT
:PU,@DT
:**********************
:** Load Program INIT *
:RU,MLLDR,#INIT
:PU,INIT
:SP,INIT
:PU,@INIT
:**********************
:** Load Program MTRP *
:OF,MTRP
:RU,MLLDR,#MTRP
:PU,MTRP
:SP,MTRP
:PU,@MTRP
:**********************
:** Load Program PLRP *
:OF,PLRP
:RU,MLLDR,#PLRP
:PU,PLRP
:SP,PLRP
:PU,@PLRP
:**********************
:** Load Program SR   *
:OF,SR
:RU,MLLDR,#SR
:PU,SR
:SP,SR
:PU,@SR
:TR
```

Figure 30. TRMLDR - Procedure file for loading segmented programs.

```
* RU,SGMTR,@MTRP,#XX::10,28,MTRP,M  *  2:55 PM  MON., 22  FEB., 1988
SH,SHAR1
SZ,30
LI,@MTRP
LI,$PLDH2
OP,EM
OP,BP
* *TOTAL PROGRAM SIZE IN DECIMAL    34322  *SGMTR:   3 NODES CREATED
M
NA,MTRP
NA,PAS.1
NA,PAS.2
NA,PAS.STOP
.
.
M.1
NA,UPDATE_CURSOR
NA,GET_ANSWER
NA,UPDATE_DISPLAYS
NA,SET_DATA_FRAME
NA,PAS.CLOSEFILE
NA,PAS.CDSCONFLICT
.
.
M.2
NA,PM_INIT
NA,PROCESS_RESPONSE
NA,UPDATE_US
NA,SET_DATA_FRAME
NA,PAS.CLOSEFILE
NA,PAS.CDSCONFLICT
END
```

Figure 31. Sample loader directive file #MTRP.

As one can see from Figure 31, several modules, "**SET_DATA_FRAME**" through "**PAS.NONCDS**", are repeated in the two nodes M.1 and M.2. These modules are repeated because they are used in both paths; repetition of modules requires more and more of the scarce memory space needed for a memory resident program. In general, the more nodes there are, the more repetition of modules there is and the greater the memory requirements for these memory-resident programs (refer to RTE-6/VM Loader Reference Manual, p. 6-3, NOTE at bottom of page). For example, the listing in Figure 31 shows the size of the program MTRP as 34,322 machine words, but the partition reserved for the exclusive use of the program MTRP is set at 49,000 words. The difference is primarily due to the repetition (multiple copies) of various modules because they are used in several segments on independent paths.

Careful hand modification of the loader directive modules #MTRP and #PLRP can reduce the memory requirements for the programs MTRP and PLRP from 49 pages to approximately 40 pages each. This frees 18 pages of badly needed memory space for other uses such as EMA. Appendix G presents an example of how to improve the resource requirements and load time of segmented programs by manually changing the loader directive files.

NOTE

Notice in line 1 of Figure 31 that the segmenter was told there were 28 pages available for the longest path and that, two lines down, the loader is told to allow 30 pages (SZ,30) for the longest path. This ensures that there will be at least two pages of scratch memory available for the operating system to manage the stack and, more importantly, the memory-resident segments of these programs. This is done for both the MTRP and PLRP programs. When these two programs were converted from disc- to memory-residency, the lack of adequate scratch space resulted in the run-time error "HEAP - STACK COLLISION". Refer to the RTE6/VM Loader Reference Manual, p. 6-3, paragraph 2 and p. 8-2 under the Pascal heading for more detail. Most of the other segmented programs are segmented with the path length set to 29 pages and then loaded with the same 30-page size specification. The exceptions are the programs DT and SR, which are segmented with a path limit of 17 pages. This limitation allows room for loading the DS routines that were not included in the segmentation process.

8.2.6 On-Line Driver Replacement

As mentioned in Section 6.3, one special-purpose driver was created by the TRAMCON developers to support the TRAMCON remote units. To aid in debugging this driver without having to regenerate the system each time a change is made to the driver, HP has supplied an On-Line driver replacement package consisting of two programs, DRREL and DRRPL. This section describes how to use these programs to replace and test changes to the remote unit interface driver DVA76. The same could be used for DVA77, but since these drivers are essentially the same, the debugging can be done using DVA76 and the results applied to DVA77.

The two programs DRREL and DRRPL perform the driver relocation and the actual driver replacement, respectively. Before these two driver replacement utilities can be used, they must be installed in your system by using the program LINK as shown in Figure 32.

The two procedure files listed in Figure 32 were delivered with the TRAMCON software development system as disc files (DRREL and (DRRPL, respectively. The relocatable files %DRREL and %DRRPL are found on the system software tape delivered by HP.

Procedure File (DRREL

```
:**
:**   INSTALL ON-LINE DRIVER RELOCATION UTILITY - DRREL
:**
:OF,DRREL
:RU,LINK,%DRREL,DRREL::2
:**
:TR
```

Procedure File (DRRPL

```
:**
:**   INSTALL ON-LINE DRIVER REPLACEMENT UTILITY - DRRPL
:**
:OF,DRRPL
:RU,LINK,%DRRPL,DRRPL::2,+SZ:21
:**
:TR
```

Figure 32. Installation of DRREL and DRRPL.

Note that DRRPL must be sized to 21 pages (+SZ:21) to accommodate drivers of the size of DVA76. If an attempt is made to replace a driver that is too large for DRRPL to handle, the message

```
/DRRPL: DRRP 013  TABLE OVERFLOW
/DRRPL: DRRPL ABORTED
```

will be displayed. Simply relink DRRPL with a larger size specification. Now that DRRPL and DRREL are linked into the system, we are ready to use them to replace any driver on the system.

In order to relocate a new copy of a driver, the location of the original driver must be determined. The best way to determine the location of any driver is to refer to the generation listing. When the software was turned over by ITS, the generation listing file name was "TRMCN. Search the listing for a line similar to the following:

```
DVA76          (    0)  4056   6757 remote unit driver  Date: 870427
```

The above line indicates that driver DVA76 occupies memory locations 4056 to 6757 in the operating system's address space. Keep in mind that these two addresses and all other addresses specified while using the driver replacement programs are specified in OCTAL.

The following is a sample run of the driver replacement programs DRREL and DRRPL. The operator responses are underlined and all responses should be in upper case. In the following discussion and example, Octal numbers are represented by placing a "B" after the digits.

When replacing an existing driver, the current memory addresses for the existing driver must be determined. The address range for all drivers is shown by the program DRREL after the first question is answered. In the example, the driver partition is three pages long, starts at address 6000B and ends at 13777B. The answers to the next two questions (LOW LOGICAL ADDRESS? and HIGH ADDRESS?) must be in this range and may not be known exactly until the DRRPL program is run. To get the exact addresses from DRRPL, first run DRREL and specify any addresses that are within the allowable range (in the example 6000B to 13777B).

Before running DRRPL, the number of the driver partition (or physical page number of memory) containing the driver to be replaced must be determined. One way to acquire the memory page number is to search the System Generation listing as mentioned above. Another method is to examine the output of utility program LUPRN. Program DRRPL will show the actual memory space occupied by each driver in the chosen partition. In the example, the driver being replaced, DVA76, starts in location 11214B and ends one word before the next driver starts at address 12102B. Once these addresses are known, rerun DRREL and supply the proper addresses.

Sample execution of program DRREL

```
:DRREL

BASE PAGE LINKS (Y,N)? N

  DRIVER PARTITION     3 PAGES     6000 13777
            SYSTEM DRIVER AREA    24000 25621

LOW LOGICAL ADDRESS? 11214B

HIGH ADDRESS?  12102B

OUTPUT FILE? DVR

/DRREL:  RE,%DVA76
  DVA76                11214 12057  remote unit driver  Date: 880211
    CA76                 11344
    IA76                 11214

/DRREL:   EX

  ******      1  DRIVER     ******
  LOGICAL ADDRESS  11214 12057
     NO BASE PAGE
/DRREL:  FILE DVR    READY AT  2:57 PM  MON., 13  SEP., 1987
/DRREL: END
```

Figure 33. Sample execution of program DRREL.

```
:DRRPL

/DRRPL:  DR.DVR
    1  DRIVER
LOGICAL ADDRESS   11214 12057
   NO BASE PAGE

MEMORY OR PERMANENT REPLACE (ME,PE)? ME

PHYSICAL PAGE OR DRIVER PART. (PG,DR)? PG

DP STARTING PHYSICAL PAGE? 46

DRIVER PARTITION   4   <<PAGE   46>>
 EQT   4 = SC 15 TYPE 76        T=      0  X=  13 IN= 11214 CC= 11340
 EQT   5 = SC 16 TYPE 77        T=      0  X=  13 IN= 12103 CC= 12227
 EQT   7 = SC 20 TYPE 66        T=      0  X=  12 IN= 11214 CC=  6155
 EQT   8 = SC 20 TYPE 66        T=      0  X=   0 IN= 11214 CC=  6155
 EQT   9 = SC 21 TYPE 66        T=      0  X=  12 IN= 11214 CC=  6155
 EQT  10 = SC 21 TYPE 66        T=      0  X=   0 IN= 11214 CC=  6155

WARNING:  MAY BE OVERLAYING DRIVERS.  TYPE '/A' TO ABORT

DVA76
IA76    11214
CA76    11344

EQT NUMBER?  4

DMA (Y,N)? N

AUTOMATIC OUTPUT BUFFERING (Y,N)? N

SELECT CODE? 15B

TIME OUT INTERVAL? 0

EQT NUMBER (/E TO END)?  /E
```

Figure 34. Sample execution of program DRRPL.

```
INTERRUPT TABLE CHANGES:
 S.C.     INTERRUPT TABLE     TRAP CELL  (MEMORY)
  15           EQT    4          105356

INTERRUPT TABLE MODIFY:

SELECT CODE (/E TO END)? /E

SUMMARY OF SYSTEM CHANGES:
*********************************************************************
DRIVER PARTITION   4   <<PAGE   46>>
LOGICAL ADDRESS  11214 12057
   NO BASE PAGE

EQT CHANGES:
 EQT   4 = SC 15 TYPE 76        T=      0  X=  13 IN= 11214 CC= 12057

INTERRUPT TABLE CHANGES:
 S.C.    INTERRUPT TABLE   TRAP CELL  (MEMORY)
  15           EQT    4          105356
*********************************************************************

***************************************************************
***************************************************************

          WARNING!    WARNING!    WARNING!    WARNING!

        THIS CAN CRASH YOUR SYSTEM!  THE SYSTEM SHOULD BE
        INACTIVE.  EQTS TO BE REPLACED MUST NOT HAVE
        REQUESTS PENDING.

***************************************************************
***************************************************************

MEMORY REPLACE READY (Y,N)? Y
```

Figure 34. (cont.)

8.3 Miscellaneous FMGR Procedure Files and Program Command Files

This subsection discusses FMGR procedure files and other program command files that are not mentioned previously in this section because they are not used to accomplish any of the major software development functions. Nevertheless, these procedure and command files have proven very useful and should continue to do so for future software maintenance.

All these files are type 4 text files residing on the disc cartridge 10 and are created and maintained by the text editor EDIT. The file listed in Figure 35 is used as input to the program FC to save all the TRAMCON executable files on magnetic tape. Program FC is directed to this file for its instructions by specifying the file name in the run-string as in the command "**RU,FC,TR,TRFC**". The TR parameter is commonly used to indicate that the program, in this case FC, is instructed to transfer its command processing control to the file whose name follows, in this case "**TRFC**".

```
DE,-2,-8
GR
CO,AL
CO,AUTOR
CO,CC
CO,CF
CO,CMMD
CO,CN
CO,CO
CO,CR
CO,DT
CO,ED
CO,HI
CO,HR
CO,INIT
CO,KYBRD
CO,LO
CO,LOF
CO,LON
CO,LS
CO,MA
CO,ME
CO,MS
CO,MSG
CO,MTRP
CO,PA
CO,PC
CO,PF
CO,PH
CO,PLRP
CO,PM
CO,POLL
CO,PR
CO,SC
CO,SE
CO,SETCL
CO,SETDT
CO,SETVE
CO,SETCR
```

Figure 35. FC command file - TRFC.

```
CO,SI
CO,SR
CO,SS
CO,SW
CO,TROFF
CO,TS
CO,UP
CO,US
CO,WZ
CO,X
CO,ARPTR
CO,RMASTE
CO,TIMSET
CO,TIMPAS
CO,BROADC
CO,CHECKT
EG
EX
```

Figure 35. (cont.)

By copying the executable modules to tape using FC, the working copy of any individual program can be restored to disc (in case problems were encountered in program development) without having to do the lengthy restoral of the entire disc.

The FMGR procedure file RN, listed in Figure 36, is very useful for changing the Configuration data base by hand. Originally created to help develop and debug the TRAMCON "CO" command, this procedure file is still useful. Using this file, one can change from data base to data base much faster than using the CO command.

```
:************************************
:** Rename NEW ")" to TEMPORARY "*" *
:************************************
:RN,)DICT:2810,*DICT
:RN,)NET:2810,*NET
:RN,)LINKS:2810,*LINKS
:RN,)MAST:2810,*MAST
:RN,)LINK:2810,*LINK
:RN,)REMOT:2810,*REMOT
:RN,)SEG:2810,*SEG
:RN,)TRUNK:2810,*TRUNK
:RN,)EQT:2810,*EQT
:RN,)CRT:2810,*CRT
```

Figure 36. RN - Configuration data files rename procedure.

```
:RN,)SITE:2810,*SITE
:RN,)DINIT:2810,*DINIT
:***********************************
:** Rename CURRENT "(" to NEW ")"   *
:***********************************
:RN,(DICT:2810,)DICT
:RN,(NET:2810,)NET
:RN,(LINKS:2810,)LINKS
:RN,(MAST:2810,)MAST
:RN,(LINK:2810,)LINK
:RN,(REMOT:2810,)REMOT
:RN,(SEG:2810,)SEG
:RN,(TRUNK:2810,)TRUNK
:RN,(EQT:2810,)EQT
:RN,(CRT:2810,)CRT
:RN,(SITE:2810,)SITE
:RN,(DINIT:2810,)DINIT
:***********************************
:** Rename OLD "^" to CURRENT "("   *
:***********************************
:RN,^DICT:2810,(DICT
:RN,^NET:2810,(NET
:RN,^LINKS:2810,(LINKS
:RN,^MAST:2810,(MAST
:RN,^LINK:2810,(LINK
:RN,^REMOT:2810,(REMOT
:RN,^SEG:2810,(SEG
:RN,^TRUNK:2810,(TRUNK
:RN,^EQT:2810,(EQT
:RN,^CRT:2810,(CRT
:RN,^SITE:2810,(SITE
:RN,^DINIT:2810,(DINIT
:***********************************
:** Rename TEMPORARY "*" to OLD "^" *
:***********************************
:RN,*DICT:2810,^DICT
:RN,*NET:2810,^NET
:RN,*LINKS:2810,^LINKS
:RN,*MAST:2810,^MAST
:RN,*LINK:2810,^LINK
:RN,*REMOT:2810,^REMOT
:RN,*SEG:2810,^SEG
:RN,*TRUNK:2810,^TRUNK
:RN,*EQT:2810,^EQT
:RN,*CRT:2810,^CRT
:RN,*SITE:2810,^SITE
:RN,*DINIT:2810,^DINIT
```

Figure 36. (cont.)

The procedure RN shown in Figure 36 renames all three sets of data base files in a round robin fashion. The net effect is to restore the old data base as the current data base. To incorporate a new data base by hand, the procedure has to be modified to rename the files in the following way:

1. Purge TEMPORARY "*" if they exist
2. Rename OLD "^" to TEMPORARY "*"
3. Rename CURRENT "(" to OLD "^"
4. Rename NEW ")" to CURRENT "("
5. Rename TEMPORARY "*" to NEW ")"

Both manual renaming procedures discussed above require that the data base files whose names begin with "*" are not already in existence. If they do exist, they must be purged before these procedures are executed.

A new data base consisting of the 12 files whose names begin with ")" is distributed to the field on tape cassette in File Copy (FC) format. The RE procedure file shown in Figure 37 is required to be on disc LU 10 of every TRAMCON field system and is used by field personnel to copy these new files from tape to disc. The statement "**:RU,FC,CO,-8,,BDV**" runs the program FC and instructs it to COpy the entire contents of the tape (LU = 8) to the same disc LU from which they were copied to tape, replacing any files of the same name (D) and Verify the results (V).

The line in procedure RE above that reads "**::)MISC**" was included to allow software developers or data base distributors a means by which they could accomplish any emergency function or system modification to a field system. The line instructs FMGR to transfer control to procedure)MISC, which is required to be on cartridge 10 of every TRAMCON master system and is listed in Figure 38.

```
:SV,4,,IH
:TE, **********************************************
:TE, ***  Installing NEW Configuration Data      ***
:TE, **********************************************
:RU,FC,CO,-8,,BDV
::)MISC
```

Figure 37. RE - New configuration data base REplacement.

```
:TE,*******************************************************
:TE,**** MISCELLANEOUS FILE FOR EXECUTING UTILITY     *****
:TE,**** PROGRAMS AS NEEDED                           *****
:TE,*******************************************************
:PU,DSINIT::2
:RU,SETDT
::
```

Figure 38. Current contents of file)MISC.

The functions performed in file)MISC in Figure 38 are not so much to fix an emergency situation but rather to allow the acceptance of a new Configuration data base.

NOTE

The first statement, "PU,DSINIT::2", removes a disc file that is no longer used because of the change to the Configurator which automatically generate this file for each new data base. This change is part of Version 1.8. Starting with Version 1.8, any software tapes that are sent to the field should have this statement removed from file)MISC and the file DSINIT removed from LU 2.

By running program SETDT, the flag "**configuration_flag**" in the (DATE file is set to indicate that there is a new data base available for use. Refer to Section 14 of this manual for further discussion of data base distribution.

8.4 Source Code Structure and Writing Conventions

Each compiler or assembler requires certain directive statements to be present in the source file so that it can determine what options to use in compiling or assembling a particular module. The first convention to mention is the disc file naming convention. All stand-alone source module file names begin with the letter "&". A stand-alone module is one that will compile or assemble by itself, rather than having to be INCLUDED in another module to be compiled or assembled. Most of the source code modules are the stand-alone type. The few INCLUDE modules involved in TRAMCON are discussed in Section 11 of this manual.

8.4.1 Source Code Structure

Figure 39 shows a sample Pascal source file for a typical TRAMCON program. The Pascal compiler directives (referred to in the manual as "**OPTIONS**") are discussed in the Pascal/1000 Reference Manual, Appendix D. Any directives appearing in the TRAMCON source files are there because the non-default value of that OPTION is to be used. The first directive in the source file is the $PASCAL directive, which is discussed on page D-10 of the Pascal manual. Although this directive is optional, it is specified in all TRAMCON program modules to produce the program identification string that appears on load maps. The first line of each program module also contains the HEAP directive, which tells the compiler that the program's HEAP is to be placed in EMA and, therefore, requires two-word addressing. This EMA HEAP (HEAP 2) is the vehicle by which programs running under RTE-6/VM can share data, and therefore, most TRAMCON On-Line programs use the EMA HEAP. The last directive on the first line is the HEAPPARMS directive explained on page D-6 of the Pascal manual. By setting HEAPPARMS OFF for most of the TRAMCON code, program execution time for handling VAR parameters is essentially cut in half because the compiler generates one-word addresses rather than two-word HEAP 2 addresses.

The second line of directives eliminates the need for additional stack space, stack handling code, and range checking code. There is only one recursive routine ("**evaluate_node**" in $MPLIB) in all the TRAMCON On-Line software, so all program modules set RECURSIVE OFF at the beginning. RECURSIVE is turned on and off around that routine in $MPLIB. Range checking was considered to be a software development tool; therefore, the RANGE OFF directive was not placed in any source module until that module was considered to be in production form. Since eliminating the range checking code results in almost 10% savings in space and execution time, the decision was to use this directive for the production software. The TRAMCON software system is a closed system. This means that, once the modules have been tested and placed into production, there can be NO expectation of program failure due to values exceeding their designated bounds. This does not mean range violations will never occur. Rather, if it does occur, a coding error has been discovered that must be fixed. The contents of the Configuration data base cannot cause a software failure. All dynamic data files are fixed record length and fixed in overall length. If written properly, the software will never attempt to access these data files out-of-bounds. Any errors occurring in this theoretical system will be the result of hardware failure. The library modules &TRLIB and &MPLIB require the SUBPROGRAM directive, which must be placed anywhere before the PROGRAM statement.

The only other compiler directives that occur in any TRAMCON source modules beyond these first two lines of directives are the INCLUDE, RECURSIVE, HEAPPARMS, and DIRECT directives. By far, the most commonly used directive of these four is the INCLUDE directive. The INCLUDE directive is used to include the TYPE and CONST definitions from module [RECR3 into the compilation of almost every program in the TRAMCON system. Also included in almost every TRAMCON program is the global VAR declaration block [TRVAR. A few programs include the VAR declaration block [MPVAR. Only the program DT includes the module [DTVAR rather than [TRVAR. Refer to Section 11.3 of this manual for a detailed discussion of these INCLUDE modules.

The most recent INCLUDE module is [EXTNT, which is a small VAR section followed by a few procedure declarations. Since this module contains a VAR section followed by some procedures, it must be included immediately following any global VAR and CONST sections and immediately before any level 0 procedure declarations within a program module. Figure 39 shows the proper placement of the INCLUDE statement for [EXTNT. Refer to Section 6.4 for a discussion of the extended or LOGICAL remote unit feature. The RECURSIVE directive is used once in the entire TRAMCON system to bracket the routine "**evaluate_node**" in $MPLIB.

NOTE

There is a discussion of the SUBPROGRAM and LIBRARY directives on page D-14 of the Pascal Reference Manual. After reading the information under the SUBPROGRAM heading on page D-14, one would expect to use both the LIBRARY and SUBPROGRAM directives to compile the library modules &TRLIB and &MPLIB. If the LIBRARY directive is specified, the compiler will display something similar to the following on the screen:

```
     1    0  : $PASCAL 'TRAMCON Library, Ver. DEV', SUBPROG
S OFF$
0  *** Warning:  This feature is HP-1000 Pascal

     2    0  : $RECURSIVE OFF, RANGE OFF, LIBRARY$
1  *** Warning:  Option not recognized: LIBRARY

Pascal :  0  errors and 2 warnings in file &TRLIB
Pascal :  Macro scheduled
Macro  :  Ran out of scratch file space for swapping data to disc
Macro  :  No  errors total
Pascal :  Assembly source kept in file ^TRLIB
```

Not only is the LIBRARY option not recognized, but the operation of the Macro assembler is affected. Also, it appears that the entire contents are not loaded if the LIBRARY directive is omitted as stated on page D-14. On the other hand, the SUBPROGRAM directive is required for the library modules even though it results in a warning message.

NOTE

The DIRECT option is used to define the type of calling sequence used by a select few routines from the relocatable library supplied by HP. One of those routines is IAND, which is defined by the source line

```
FUNCTION iand (val, mask: INT): INT $DIRECT$ ; EXTERNAL;
```

Almost every routine called by the TRAMCON system uses the normal indirect calling sequence. Only a select few use the direct method. Discovering which calling sequence is used by each routine in the relocatable library takes a little detective work. As an example to illustrate how to make this determination, the reader is referred to the Relocatable Libraries Reference Manual, pp. 3-42. This page describes the routine IAND and illustrates the calling sequence used by this routine with a few lines of assembly code:

```
JSB IAND
DEF i
DEF j
Return (Result in A)
```

The above four lines of assembly code illustrate the direct calling sequence, while the following lines, shown for the routine IDIM on the facing page 3-42, illustrate the indirect method:

```
JSB IDIM
DEF *+3
DEF i
DEF j
Return (Result in A)
```

The single difference between the two calling sequences shown above is the extra return address DEF *+3 in the indirect sequence. This method allows for optional parameters. That is, routines using this calling sequence can be called with more or fewer parameters than expected and still have the proper return address. Routines like IAND, which always have the same number of parameters specified when they are called, can use the direct method. The programer must be a bit cautious when using routines from the relocatable library. There is a discussion of the direct option in the Pascal/1000 Reference Manual, pp. 8-49 - 8-50. Here it explains that a time savings can be realized for frequently-called routines by specifying the direct option. Of course, we do not have a choice for routines written elsewhere or for which we do not have the source code, but even for the user written routines, the use of DIRECT was not explored. If some tailoring for speed is desired in the future, this is one possible consideration.

```
$PASCAL 'Remote Alarm-Status display' , HEAP 2 ,HEAPPARMS OFF$
$RECURSIVE OFF , RANGE OFF$
PROGRAM al;                   {<870806.1142>}
$INCLUDE '[RECR3'$
$INCLUDE '[TRVAR'$
    l,alord,al_type,match,snl_len,sn2_len,msg_displayed,nrecs,
      sav_archidx,sav_extidx,arch_idx,arch_end,recs_to_print,
      recs_printed,prevrem,base_rec,save_next: INT;
    arch_records:ARRAY[1..max_archive_record] OF parm_array;
    .

    .
$INCLUDE '[EXTNT'$
PROCEDURE allocate_EMA (stack_size:INT;VAR id: byte;incrt:BOOLEAN);EXTERNAL;

PROCEDURE get_parms $ALIAS 'Pas.NumericParms'$
                    (VAR parms:parm_array);EXTERNAL;

PROCEDURE jtime (VAR time_alf: time_str); EXTERNAL;

PROCEDURE arch_fkeys ;

BEGIN {arch_fkeys}
write(outunit,esc,'&f2a3k8DNext Rec3',esc,'&f2a5k6D PRINT5',
             esc,'&f2a4k8DPrev Rec4',esc,'&f2a6k8DTimeDate6',
             esc,'&f2a7k8DRecord #7',esc,'&f2a8k6D  LIST8',esc, '*m3M')
END;  {arch_fkeys}
.

.
PROCEDURE search_archives (reset_idx: BOOLEAN);

VAR i,j,k,l,retry,save_nxt,last_rec: INT;

BEGIN {search_archives} nrecs := 0; remoteord := parms[2];
rem_status_ptr := heap^.segment_status[segord].remote_status[remoteord];
WITH rem_status_ptr^ DO
  BEGIN save_nxt := next_archive_record;
  IF save_nxt < 0 THEN
    WHILE save_nxt < 0 DO
      IF next_archive_record = -32002 THEN
        BEGIN pause(12,0,2,0,-1); save_nxt := next_archive_record;
        IF save_nxt = -32002 THEN
          IF retry = 0 THEN
            BEGIN save_nxt := 0; remoteord := max_remotes_per_segment;
            nrecs := 0; display_msg(4)
            END
          ELSE retry := retry + 1
        END
      ELSE
      IF next_archive_record = -32001 THEN
```

Figure 39. Sample Pascal source file.

```
        BEGIN save_nxt := 0; display_msg(2); nrecs := 0;
        remoteord := max_remotes_per_segment
        END
      ELSE
      IF (next_archive_record < 0) AND (next_archive_record > -32000) THEN
        save_nxt := abs(next_archive_record);
  IF remoteord < max_remotes_per_segment THEN
    BEGIN i := 1; nrecs := max_archive_record; j := save_nxt;
    IF save_nxt > 0 THEN WITH archive_rec[0].arch_rcd DO
      WHILE i < max_archive_record+1 DO
        BEGIN readdir(archive_file,j,archive_rec[0]);
        IF arch_year = 0 THEN
          BEGIN
          IF j = base_rec THEN nrecs := 0
          ELSE
            BEGIN nrecs := j - base_rec;
            IF nrecs > 50 THEN display_msg(1); k := base_rec;
            FOR l := 1 TO nrecs DO
              BEGIN readdir(archive_file,k,archive_rec[0]);
              arch_yrs[l] := arch_year; k := k+1
              END
            END;
          i := max_archive_record+1
          END
        ELSE
        .
        .
        END {WHILE i<max_archive_record+1}
    END
  END;
.
.
REPEAT  heap^.current_crt[crtord].miscl := nill;
  REPEAT get_category; display_category UNTIL soft_key <> soft3;
UNTIL soft_key <> soft4;
.
.
END;  {search_archives}

BEGIN {al} get_parms(parms); allocate_EMA(0, id, TRUE);
redo_hdg := TRUE; msg_displayed := 0; tab_set := FALSE;
WITH heap^.current_crt[crtord] DO BEGIN max_dsp_ln:=20; locked_ln:=2 END;
IF print_it THEN print_done
END.  {al}
```

Figure 39. (cont.)

8.4.2 Writing Conventions

The following discussion concerns the rules set up to govern the format of the source code itself. The delimiters "{" and "}" are used for all comments rather than "**(***" and "***)**". The time-stamp, which is maintained by the HP supplied source code editor EDIT, is placed on the program statement line and is enclosed in comment brackets.

The BEGIN and END identifiers that demark the main body of any procedure, function, or program are labeled with the name of the procedure, function, or program enclosed in comment brackets. In the example shown in Figure 39, the program name is "**al**". The body of the program can be easily identified by finding the "**BEGIN {al}**" and "**END. {al}**" strings. This begin/end labeling scheme becomes very useful in matching begins and ends as the included code grows large.

Along with marking the BEGIN/END pairs, indentation is used to help the programer visualize blocking created by use of compound statements within the program. Any statement that is at the outermost level of each procedure, function, or program is placed at the left margin. This includes all the procedure, function, and program headings and the main bodies. Notice that this rule implies that indentation is NOT used to indicate program nesting levels. Any statement that either is blocked within the previous statement or is a continuation of the same statement is indented two spaces. The ELSE clause of an IF statement is an exception to this rule because it is placed at the same indentation as its associated IF. In Figure 39 this indentation scheme is superbly demonstrated by the procedure "**search_archives**". Any statement bracketed by a BEGIN/END pair is placed at the same indentation as the BEGIN/END brackets so that the indentation does not rapidly get out of hand. This method of indenting turned out to be just as readable as indenting all the included statements two spaces more than the BEGIN/END.

If a statement, including BEGIN/ENDs and REPEAT-UNTILs, could be placed entirely on one line, it was. This is illustrated by the third to last line in Figure 39, which shows an entire compound statement on the same line. It is an advantage to be able to view a maximum amount of code on a single screen. To make this possible, often, several simple statements were grouped on the same line.

At all times, no line exceeded 80 columns, which avoided difficult-to-read line wraparound on the screen.

Unlike the BEGIN-END, the statements included within the REPEAT-UNTIL and the CASE-END are indented two spaces from the REPEAT-UNTIL or CASE-END.

The Pascal/1000 compiler does not distinguish between upper- and lowercase letters within identifiers (refer to Pascal/1000 Reference Manual, p. 2-5). Therefore, the rule for the use of upper- and lowercase in the TRAMCON software was based on cosmetic considerations and was made to make the source code more readable. All Pascal reserved words used in the TRAMCON software and listed in the Pascal/1000 Reference Manual, p. 2-4, are written using all capital letters. Also, all capital letters are used for the predefined identifiers listed on page 2-7 of the Pascal Reference Manual. Examples of reserved words in Figure 39 are BEGIN, END, REPEAT, and PROCEDURE. Examples of predefined identifiers in Figure 39 are TRUE, FALSE, and BOOLEAN. Most of the other identifiers are composed of lowercase letters with liberal use of the underscore character. The one exception is the user-defined type **INT**, which is spelled with all caps because it is so close to being a basic type.

Most of the user-defined TYPEs and CONSTANTs are grouped into one INCLUDE module called [RECR3, which is detailed in Section 11.1 of this manual. A programer attempting to modify or add to this code should consult this module for identifiers. These identifiers can be used for commonly-used constants and types, such as the one-word integer constant "-1", which has the name "**nill**" or the one-word integer type INT.

The last software-writing style item mentioned here is the prolific use of the WITH statement throughout the TRAMCON On-Line software. There is a discussion about this in the Pascal/1000 Reference Manual, p. 8-47, which says that using the WITH statement can increase execution efficiency and reduce source code repetition. The TRAMCON On-Line software deals primarily with information stored in a hierarchical structure in the shared two-word addressable area called EMA. With two-word addressing, savings on de-referencing are even more important. Also, to specify an item down to the most-nested level in the HEAP sometimes takes more than 80 characters. Repeating these long identifiers makes the code very difficult to read.

The only drawback to using the WITH statement is the difficulty in pinpointing the location or actual full identity of some items in the code when troubleshooting or changing the code. With this in mind, an attempt was made to give unique identifiers for everything, including fields within records, so that an identifier that is specified partly in a WITH statement, would not be confused with a simple identifier with the same spelling.

Another policy decision that had to be made concerned the indexing of arrays. In PASCAL, there are no restrictions on values for array indices, but the most common choices for the range of numeric indices are "**0..n-1**" and "**1..n**". There are advantages and disadvantages to both ranges. There are machine instructions to TEST AND BRANCH IF ZERO, which can be used to generate more efficient code for LOOPS when indexing through arrays whose indices begin at zero. The readability of the source code is improved if the "**1..n**" indexing is used because the "**-1**" does not have to be entered. We chose the "**0..n-1**" indexing range, and for uniformity we strongly urge that this convention be followed.

9. TERMINAL I/O MANAGEMENT

A minimum of one and a maximum of four terminal display devices can be connected to a TRAMCON master at one time. The type of terminal devices supported by TRAMCON is currently limited to HP equipment. Because of the design of the terminal handling code, it should not be too difficult to change TRAMCON to support a wider range of terminals. Essentially, the limiting factor is the particular interface chosen for the terminal devices. The BACI interface occupies one back-plane slot yet supports only one terminal device. Even more limiting is the fact that the driver, which supports the BACI interface, uses the HP unique ENK-ACK handshaking protocol. Most of the rest of the industry uses the XON/XOFF handshake.

This is mentioned not to encourage the development of code to support new terminal devices, but to explain some of the ideas that lead to the design of the terminal support code. Currently, three models of HP terminals are supported by the TRAMCON software. They are, from oldest to newest, the HP-2647F, the HP-2627A, and the HP-2397A. The newer terminals were added because the older HP-2647F was expensive and no longer supported by HP and because the need for a stand-alone workstation connected to the TRAMCON master never materialized. These reasons alone were sufficient to justify the effort to add support for the new equipment. The color feature was an excellent by-product.

The speed with which the TRAMCON master and the terminal devices communicate is limited to a maximum of 9600 bps. This limitation is only because of the BACI interface, since the newer terminal devices are capable of speeds up to 19,200 bps. Every terminal and the BACI interface are capable of using all settings for Parity, STOP bits and DATA bits. The most common settings were chosen for TRAMCON (NO Parity, 1 STOP bit and 8 DATA bits). These settings are issued to every terminal device defined in the TRAMCON data base when the system is re-booted. If a terminal exists but is not defined in the TRAMCON data base, the TRAMCON software will not attempt to use that device.

The Configuration data base contains an array called "**crt_rec**" in the master record that includes the initial values for each terminal defined on the master. A key position in this array is the position 0. "**crt_rec[0]**" must always have a terminal description, which is assumed by the TRAMCON software to be the system console. As we will see later, there is special treatment for the system console terminal vs the other terminals, which is referred to as remote display terminals or RDTs.

There is an array called "**current_crt**" defined in the run-time HEAP, which contains the status of all terminal devices defined on the given master. This array is initialized at bootup time by the procedure "**Initialize**" in the program CMMD. Some of the information such as the terminal location is transferred from the data base to the HEAP. The system console, since it must always be defined, is assumed to be hard-wired. All other terminals defined on a master are treated as if they are operated over a modem connection. If actual modems are not being used, a null modem device must be placed in line between the BACI interface and the terminal device. Different cables are used to support the hard-wired and the modem connections. Also, each new terminal model has a different communication line connector. The cables required for each terminal type are listed in Figure 40.

Terminal Model	BACI to Terminal / BACI to Modem	Modem to Terminal	Connector Type
HP-2647F	12966-60008 / 12966-60006	13232M	264x hood
HP-2627A	12966-60014 / 12966-60006	13222N	262x 50-pin
HP-2397A	12966-60015 / 12966-60006	40242M	25-pin RS232

Figure 40. TRAMCON master - terminal communication cables.

The numbers listed in Figure 40 are HP part numbers. All cables listed in the "**BACI to Terminal/BACI to Modem**" column have the HP-1000 back-plane interface hood for the computer end. The numbers in this column begin with 12966 because that is the part number for the BACI interface. The second half of the number in the column defines the connector type of the other end of the cable. All terminal types use the same cable, 12966-60006, when connecting from the BACI interface to a modem. This cable has a standard RS232 25-pin connector on the modem end. The connectors on the terminal end of the three hard-wired, BACI-to-device cables are male versions of the connectors listed in the last column of Figure 40. All 25-pin cable ends are male, and all interface and terminal connectors are female.

Note that whereas the hard-wired connections require one cable, the modem connections require two cables. For example, the hard-wire connection of an HP-2397A terminal to an HP-1000 requires the one cable with part number 12966-60015. The modem connection for the HP-2397A requires both the 12999-60006 and the 40242M cable. In the modem connection, it must be assumed that the connection is from HP-1000 to modem, and elsewhere from modem to terminal.

The TRAMCON terminal I/O function is diagramed in Figure 41.

When the TRAMCON On-Line software is booted up, the program CMMD clones a copy of the program UP for each terminal device that is defined in the Configuration data base master record field "**crt_rec**". After scheduling each UP program, CMMD assumes that the terminal is NOT operational and calls routine "**down_crt**" ($TRLIB). Routine "**down_crt**", in turn, passes the information to the program UP via a CLASS WRITE/READ. The program UP attempts to write to the terminal. If the terminal is actually operational, UP detects this and schedules a clone of the program LO to prompt the operator to sign-on. The sign-on prompt will never time out and the TRAMCON On-Line software doesn't try to use the terminal unless someone is signed on.

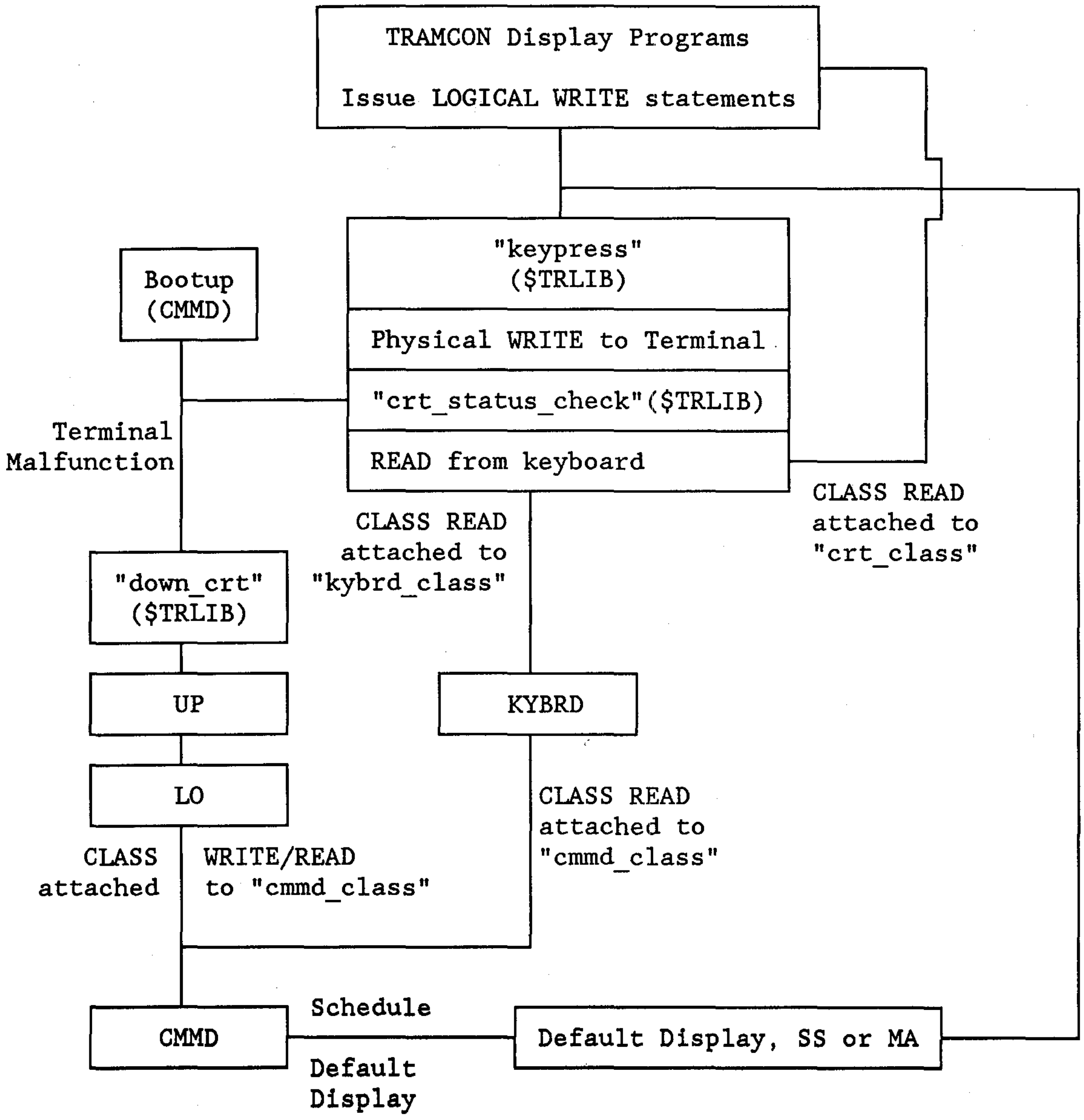

TRAMCON Display Programs
Issue LOGICAL WRITE statements
"keypress" ($TRLIB)
Physical WRITE to Terminal
"crt_status_check"($TRLIB)
READ from keyboard
Bootup (CMMD)
Terminal Malfunction
CLASS READ attached to "kybrd_class"
CLASS READ attached to "crt_class"
"down_crt" ($TRLIB)
UP
KYBRD
LO
CLASS READ attached to "cmmd_class"
CLASS attached
WRITE/READ to "cmmd_class"
CMMD
Schedule
Default Display
Default Display, SS or MA

Figure 41. TRAMCON terminal I/O processing.

When the operator successfully signs on (see Section 9.2), program LO mimics the entry of the DE command by passing the DE command to the program CMMD via a CLASS WRITE/READ request attached to CLASS number "**cmmd_class**". Program CMMD receives the DE command by issuing a CLASS GET on CLASS number "**cmmd_class**" and schedules the default display (SS or MA) for the given terminal device.

The default display program performs just like any other program that does I/O to a terminal device. Almost all the output (display) statements executed by the TRAMCON programs are LOGICAL output statements. That is, the WRITE statements that are issued cause the operating system to place data into an intermediate buffer, but NO data are actually sent to the terminal device. Once all the information going to the terminal is buffered, the display program calls routine "**keypress**" to issue a physical write (prompt) statement, flushing the buffer to the terminal device.

Only when this physical write is done can the software determine whether the terminal device is still operational. Routine "**keypress**" calls routine "**crt_status_check**" to discover whether the terminal is still functioning. If the terminal is NOT functioning, the recovery cycle begins again by calling routine "**down_crt**" (refer to Section 9.4). If the terminal is still functioning, one of two actions is taken.

In most cases, the display program's job is complete when they send all their data to the terminal display. When this is done, the call to "**keypress**" finishes by issuing a single character BINARY CLASS READ statement and attaches this read to the CLASS number "**kybrd_class**". The results of this read will be examined by program KYBRD.

The only other action taken is to issue a multiple-character, normal CLASS READ and attach the read to the CLASS number "**crt_class**". The seldom-used function is used by programs, such as DT, that interact with the operator.

The routine "**crt_status_check**" in $TRLIB issues a status request on a specified terminal and checks for a "**status3.device_down**" indication or a "**eqt4.timedout**" indication. Either of these flags is interpreted as a malfunction of the terminal device and the program UP is informed of this fact. Program CMMD issues a status request for a terminal device, after getting input from an operator at the given terminal, to see if the keyboard input request has timed out. If "**eqt4.timedout**" is true, CMMD assumes that the operator no longer wishes to enter a command, the "**Enter Command**" prompt is erased, and a new single character keyboard "**wakeup**" read request is issued. Program UP also issues a status request to a terminal device to see if the terminal is once again operational.

9.1 System Console vs Remote Display Terminal (RDT)

As mentioned earlier is this section, the software treats the system console differently than the other terminals on a given master. First of all, the HP operating system requires that a system console be present and connected to a specified back-plane slot so that the system can be booted up (see Section 15 of this manual). Throughout the system software, there are messages intended for a particular user at a particular terminal. These messages are usually sent to the designated terminal and to the system console. Some of these system console and other terminal messages are unwanted for TRAMCON operation and those unwanted messages have been eliminated by making changes to the appropriate message-generating system software modules. These changes constitute changes to the operating system as it is shipped from HP and must be remade each time the system is regenerated. These changes are discussed in Section 10.3 of this manual.

In the TRAMCON system, the system console is defined as the terminal specified in the Configuration data master record in array "**crt_rec[0]**". The global variable "**crtord**", which is global to each TRAMCON program (see Section 11.3 of this manual), controls which terminal device a given program references. To reference the system console, a program must use the value 0 for "**crtord**", which is an index into the HEAP array "**current_crt**". Terminal LU values are also computed using the value of "**crtord**".

The four LU numbers 25 through 28 have been used to refer to the four possible terminals on a TRAMCON master. LU 25 refers to the system console associated with "**crtord**" value 0. The LU values have been grouped together so that the LU number for any particular terminal can be computed, given the "**crtord**", by adding 25 to the value of "**crtord**".

Once the TRAMCON system has started, the value of "**crtord**" is passed from program to program as one of the run-time parameters using the CLASS I/O feature. TRAMCON terminal I/O is a closed loop that never loses the value of "**crtord**", provided no program terminates abnormally. If a program terminates abnormally, there is a problem that must be fixed. The TRAMCON software was designed to operate without abnormal program termination.

9.2 Logging ON/OFF Remote Display Terminal (RDT)

The TRAMCON system is capable of supporting up to three additional terminal devices per master unit. These additional terminal devices are referred to as remote display terminals or RDTs. The back-plane I/O slots are defined as shown in Figure 42 for all TRAMCON master systems. The back-plane definition does not vary from master to master so that the operating system on every master is capable of supporting the same number of terminal devices.

"crt_ptr" Ordinal	Select Code	Interface	Driver	Logical Unit(s)	Equipment Number
0	13	Terminal (System Console)	DVX05	1,25	2
1	14	Terminal (Segment Console)	DVX05	26	3
2	17	Terminal (RDT)	DVX05	27	6
	23	Unused	-----	--	--
	24	Unused	-----	--	--
3	25	Terminal	DVX05	28	13

Figure 42. TRAMCON master computer terminal device back-plane assignments.

NOTE

Slots 23 and 24 are included in Figure 42 because these slots have recently become available due to the replacement of the 7900E magnetic tape drive with the streaming tape drive that is built into the disc drive.

The second factor necessary to use the terminals is not uniform for every master. That is, each terminal device must be defined in the Configuration data base master record. Each master computer has its own unique "**master_record**" that contains an array, "**crt_ptr**", which points to a CRT record for each terminal defined for use on this master. If an entry is made in the array "**crt_ptr**", the corresponding hardware interface (BACI card) must be located in the proper slot. The correspondence between the I/O slots and the elements of array "**crt_ptr**" is fixed and is shown in Figure 42. The terminal defined in "**crt_ptr[0]**" is always interpreted by the software to be the system console and always refers to the interface located in slot 13.

Any terminal device beyond the system console is optional and can be operated with a hard-wire, NULL modem, or modem connection. If the terminal device is defined in the data base and the hardware interface exists in the master back-plane, the TRAMCON On-Line software will attempt to communicate with the device starting at system bootup.

At bootup and at any future time when no one is logged on to a particular terminal device, the program LO waits for a single key (any key) to be pressed. When any key is pressed, program LO displays the prompt

Welcome to TRAMCON at xxxxxxxxxx, yyyyyyyyy

Please sign on:

The field "**xxxxxxxxxx, yyyyyyyyyy**" is the site name and country name found in the data base master record field "**site_ptr**" which, in turn, points to a data base SITE record with fields "**site_name**" and "**country**".

At this point the operator is allowed to type any character string. The first 18 characters entered will be used as the operator identification and will be displayed in the upper right-hand corner of most displays.

After the operator signs on, the program LO prompts further for the password as follows:

Enter Password

Currently, the password is fixed to be the two ASCII characters "**tr**". If the operator answers "**tr**" <RETURN> the program LO will determine which terminal type is installed at this location by reading the ID from the terminal itself. Knowing the terminal type, the graphics grid size is set and the Segment Map coordinates will be calculated. Having finished this, the program LO issues a Default Display Command (DE) to the command processing program CMMD which results in the Default display for this terminal being displayed at the terminal.

The terminal device is now a fully functional TRAMCON terminal and will remain so until it breaks or the operator logs off by entering the ST command.

9.3 Terminal Configuration

Although technically a dumb terminal, each of the HP terminal devices supported by TRAMCON software offers many operator-selectable options that allow the device to adapt to most variations in communication protocol. These settings are maintained locally at the terminal in battery-backed RAM. Since TRAMCON uses these HP terminals with HP computers, most of the factory settings can be utilized. The few settings that must be changed from the factory setting are highlighted in the following subsections. The procedure for viewing and updating the settings is also specified below. The 2397A terminal is much more elaborate than the 2627A, but most pages of settings can be used exactly as set by the factory and are therefore not listed in the following subsections. Remember, these settings are local to the terminal device and viewing or changing these values is a local operation and has no direct effect on the TRAMCON master computer.

The various options and their current settings are organized into PAGES or SCREENS of related items. Once the key-pressing procedures listed below are executed, the desired information will be displayed along with the function key label line at the bottom. The function keys are used to change the items displayed. The TAB key will expedite movement from item to item. This data changing is done in SCREEN mode. That is, none of the changes are permanently recorded in the terminal RAM until the <SAVE CONFIG> key is pressed.

9.3.1 HP-2397A

Figures 43 through 45 show the three configuration pages for the HP-2397A terminal, which include changes to the factory settings for TRAMCON operation. The settings necessary for TRAMCON operation are shown.

Figure 43 is the TERMINAL CONFIGURATION page for the 2397A terminal. This page can be viewed and changed by the following:

1. Press <System> key
2. Press <config keys> key (f8)
3. Press <terminal config> (f5)

```
                    TERMINAL CONFIGURATION

 RETURN Def cr    RETURN=ENTER  No                  Tab = Spaces  No

 Local Echo OFF     Caps Lock OFF    Start Column   1    ASCII 8 Bits  YES


XmitFnctn(A)  No        SPOW(B)  No    InhEolWrp(C)  No    Line/Page(D) Line

InhHndShk(G)  No    Inh DC2(H)  No    Auto Term(J)  No    ClearTerm(K)  No

InhSlfTst(L)  No                        Esc Xfer(N)  No    InhDcTest(W)  No

                 Field Separator us    Alternate Set  Line(B)

                Block Terminator rs         Transmit    All Fields
```

Figure 43. Terminal configuration page for 2397A.

```
     REMOTE DATACOMM Full Duplex Hardwired CONFIGURATION        Port 1

BaudRate   9600     Parity/DataBits None/8 Check Parity  No

Asterisk  Off            Stop Bits 1            EnkAck  Yes

  TR(CD) Hi                 SR(CH) Lo

RecvPace    None          Break Time 160      RR(CF)Recv  No

XmitPace    None      STOP Function Xon/Xoff CS(CB)Xmit  No    DM(CC)Xmit  No
```

Figure 44. Remote datacomm configuration page for 2397A.

Figure 44 is the REMOTE DATACOMM CONFIGURATION page for the 2397A terminal.
This page can be viewed and changed by the following:

1. Press <System> key
2. Press <config keys> key (f8)
3. Press <datacomm config> (f5)

```
                         GLOBAL CONFIGURATION

    Click Off                                    Terminal Id 2397A

     Bell  On            Display off     never      Language   ENGLISH

                             ALPHA WINDOW
Term Mode  HP                                Inverse Background  No

  Columns 160                                    Cursor Type Line

                            GRAPHICS WINDOW
Resolution 640x400                               Graph Compat   Off
```

Figure 45. Global configuration page for 2397A.

Figure 45 is the GLOBAL CONFIGURATION page for the 2397A terminal. This page can be viewed and changed by the following:

1. Press <System> key
2. Press <config keys> key (f8)
3. Press <global config> (f5)

9.3.2 HP-2627A

Figure 46 shows the three configuration pages for the HP-2627A terminal and the settings necessary for TRAMCON operation.

9.4 Handling DOWNED Terminals

It is assumed that at any time during TRAMCON operation, the terminal devices other than the system console could break or cease to communicate with the operator. It is also assumed that these additional terminal devices may not be available to be installed when the master is installed or that an additional terminal may be configured in at some future date for a given operational master. The software has been designed to operate normally even if any or all of the extra terminals are not operational. In order to prevent critical TRAMCON programs from being indefinitely suspended trying to communicate with a broken (possibly nonexistent) terminal device, the small program UP was designed to constantly try to access a broken device until it responds. Figure 47 is a block diagram showing the processes involved in maintaining the terminal devices.

```
                           TERMINAL CONFIGURATION

                                       FrameRate 60
   Language   USASCII
  ReturnDef cr

  LocalEcho OFF     CapsLock OFF       Start Col 01      ASCII 8 Bits YES
XmitFnctn(A)  NO       SPOW(B)  NO    InhEolWrp(C)  NO    Line/Page(D) LINE
InhHndShk(G)  NO    Inh DC2(H)  NO     Esc Xfer(N)  NO     Compat(P,Q)   OFF

FldSeparator us    BlkTerminator rs
```

```
                            DATACOMM CONFIGURATION

BaudRate  9600      Parity None
Asterisk OFF                                                EnqAck YES
                Chk Parity  NO          SR(CH) LO


RecvPace None
XmitPace None                                            CS(CB)Xmit  NO
```

```
                        EXTERNAL DEVICE CONFIGURATION

BaudRate  9600       Parity None   GraphContent  B&W   PrinterNulls 000

Printer    HP                         ImageSize x1        InvertB&W  NO

                                        SRRXmit  NO
XmitPace Xon/Xoff                     SRRInvert  NO      CS(CB)Xmit  NO
```

Figure 46. Configuration pages for the HP-2627A terminal.

Figure 47 shows that, when trouble communicating with a terminal is encountered by the software, the routine "**down_crt**" located in $TRLIB is called to try to reestablish contact with the given terminal. As a precaution, routine "**down_crt**" connects the LU for the terminal to the bit bucket (equipment 0) so that if any TRAMCON program other than UP attempts to send data to the terminal, that output will complete successfully and the program will not be suspended indefinitely waiting for the terminal to respond.

The cloned (one copy per terminal) program UP's sole function is to repeatedly attempt to communicate with the terminal until it is successful. Once it is successful, program UP reconnects the terminal LU to the proper equipment that it finds in "crt_eqt" and calls the routine "**clone_and_run**" to schedule the logon program LO.

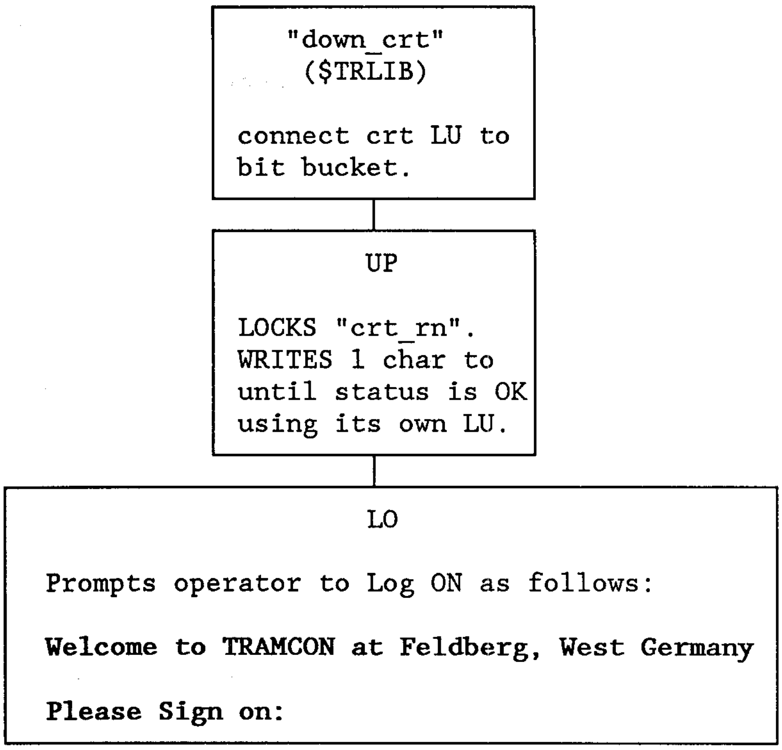

"down_crt"
($TRLIB)
connect crt LU to bit bucket.
UP
LOCKS "crt_rn".
WRITES 1 char to until status is OK using its own LU.
LO
Prompts operator to Log ON as follows:
Welcome to TRAMCON at Feldberg, West Germany
Please Sign on:

Figure 47. Handling DOWNED terminal devices.

10. SYSTEM GENERATION

Before any of the TRAMCON software could be developed, a version of the operating system had to be defined, generated, and loaded onto an HP-1000 master computer. As stated in the Introduction of this manual, the operating system chosen was RTE-6/VM. The TRAMCON software was developed and delivered using version A.85 of RTE-6/VM, which was released in the first quarter of 1985. ITS continued to receive new releases of RTE-6/VM, but found problems with several key modules such as the Pascal compiler, which led to the decision to stay with the A.85 release. Fortunately, the generation step was already improved with A.85. This step now takes approximately 12 to 15 minutes to generate a TRAMCON system rather than the previous 3 1/2 hours. By now the TRAMCON system is based on the new 7912 65-Mbyte disc drive, which replaced the old 7906 20-Mbyte drive, so the discussion here will reflect the generation of the RTE-6/VM operating system version A.85 based on a 7912 system disc drive. Also, as mentioned in the introduction to this manual, the FMGR file system was used for the development and implementation of the TRAMCON system. Therefore, none of the CI code was generated into the TRAMCON system.

The users of this manual are required to have some familiarity with the HP-1000 system including familiarity with the operating system generation process. This section describes how to generate a TRAMCON field system and not a TRAMCON development system. Unecessary details are avoided, instead concentrating on problem points and the steps unique to TRAMCON.

The first step is to ensure that the operating system modules are available for use by the generator program RT6GN. This is a difficult step when developing the first system. But, as mentioned above, it is assumed that a development system that has the operating system modules in relocatable form on the system disc drive, is already available. The next step is to develop an answer file for the generator. The generator asks a series of questions and the answer file is a disc file that supplies the answers to all those questions. Disc file naming conventions for the generation process are specified in the following rules:

1. Generation answer file begins with "AN" and indicates the capacity of the system disc in Mbytes, thus "AN65" for TRAMCON field system.
2. Generator output file is !TRMCN.
3. Generator listing is "TRMCN.

The answer file used to generate the TRAMCON system Version 1.8 is included as Appendix B of this manual.

Running the system generator is a simple process accomplished with the following FMGR commands:

```
:PU,"TRMCN::10
:PU,!TRMCN::10
:RU,RT6GN,ANTR
```

10.1 Switching to the New System

Once the generator has completed, the new system, on disc file !TRMCN, must be placed onto the system tracks of the system disc and booted up. This task is accomplished by the utility program SWTCH, which is described in detail in the RTE-6/VM System Manager's Reference Manual, Chapter 5. If the host system is the same as the target system (in most cases it will be), the SWTCH - operator interaction is shown in Figure 48.

In Figure 48, the operator responses are underlined (the responses can be in either lower- or uppercase).

```
:SWTCH
SWTCH generation file installer. Rev.2440 <850114.1607>

                ******  W A R N I N G  ******
ALL ACTIVITY MUST BE TERMINATED BEFORE SYSTEM TRANSFER PROCESS.

ENTER "!!" IN RESPONSE TO ANY QUESTION TO ABORT.

FILE NAME OF NEW RTE SYSTEM?
!TRMCN

RTE-6 SYSTEM GENERATED    9:54 AM  FRI., 12  JUNE, 1987

NEW SYSTEM I/O CONFIGURATION:

SELECT CODE 11 TBG
SELECT CODE  4 TYPE=43
SELECT CODE 11 TYPE=43
SELECT CODE 12 TYPE=33
SELECT CODE 13 TYPE= 5
SELECT CODE 14 TYPE= 5
SELECT CODE 15 TYPE=76
SELECT CODE 16 TYPE=77
SELECT CODE 17 TYPE= 5
SELECT CODE 20 TYPE=66
SELECT CODE 21 TYPE=66
SELECT CODE 22 TYPE=66
SELECT CODE 23 TYPE=66
SELECT CODE 24 TYPE= 5
SELECT CODE 25 TYPE= 5

NEW SYSTEM (LU2) SELECT CODE= 12  SUBCHANNEL=  1

# OF TRACKS    1000 ADDRESS     0
UNIT #            0 VOLUME #    0
STARTING BLOCK ADDRESS  0
# OF 128-WORD BLOCKS/TRACK    64

TARGET DISC LU FOR NEW SYSTEM?  (XX)
2
TARGET ADDRESS:UNIT:VOLUME FOR NEW SYSTEM? (X:0:0 OR " "CR)

NOW IS THE TIME TO INSERT CORRECT CARTRIDGE IN
TARGET ADDRESS:UNIT:VOLUME.  (" "CR TO CONTINUE)

SAVE FILES AT TARGET?   (Y OR N)
Y
```

Figure 48. Sample SWTCH - operator interaction.

```
LU  L-TRK   CR            LU  L-TRK   CR            LU  L-TRK   CR
 2    999    2            10   2999   10

THIS CL LOOKS REASONABLE.  IF YOU AGREE AND YOU WANT TO
SAVE IT ANSWER YES, ELSE NO.  SAVE CL?   (Y OR N)
Y
PURGE TYPE 6 FILES?   (Y OR N)
N

NO SUBCHANNEL INITIALIZATION WITH CS80

PRESENT CONFIGURATION DOESN'T PERMIT AUTO Bootup.

DISC IN HOST SYSTEM DRIVE WILL BE OVERLAID.
READY TO TRANSFER. OK TO PROCEED?
Y
```

Figure 48. (cont.)

10.2 Loading System Utilities

Only the absolutely essential software is loaded into the system at generation time. This reduces the generation processing time down to approximately 12 minutes and keeps the operating system memory and disc requirements at a minimum.

The Operating System modules, which include the I/O device drivers and the File Management (FMGR) software, are the essential portion of the TRAMCON software system. Most user-written software can be installed after the new system is generated and made operational using the SWTCH program as discussed in Section 10.1. The only user-written module included in the generation is the TRAMCON remote unit device driver DVA76.

NOTE

In addition, the On-Line loader program LOADR must also be generated in, so that the other utility programs can be loaded later. Without the LOADR program, there would be no way to load additional software after generation.

Once the system has been generated and made operational with program SWTCH, the bulk of the software is loaded into the system. Figure 49 lists the modules that are loaded On-Line after the switch to the new system has been made. Appendix C lists the FMGR procedure file *LOAD6, which can be used to load all the modules listed in Figure 49.

System Manager Utilities
RT6GN SWTCH

System Utilities
LUPRN EDITR

Program Development Utilities
DRREL DRRPL EDIT FTN7X INDXR LINDX LINK LST MACRO MLLDR PASCL
SGMTR SXREF

File System Utilities
MERGE OLDRE SCOM

Help Utilities
CMD GENIX HELP

Backup Utilities
FC TF

Diagnostic and Disc Formatting Utilities
FORMC FRMMT TVVER

Distributed System Programs
#SEND DLIST DSINF EDITR EXECM EXECW OPERM PROGL PTOPM REMAN RFAM
RSM SYSAT VCPMN

TRAMCON Segmented Programs
CF DT INIT MTRP PLRP SR

Figure 49. List of modules loaded on-line after generation.

The programs listed in Figure 49 must be reloaded even though the type 6 files were kept when the SWTCH program was run because these programs are dependant upon boundaries in the operating system, which might have changed with the new generation.

In order to load most of the software listed in Figure 49, the utility programs LINDX, INDXR, LINK, and MLLDR must be loaded into the system first and the LINK SNAP file SNAP.6 must be built. Therefore, notice that the first steps in procedure file *LOAD6, listed in Appendix C, load programs LINDX, LINK, and run program LINDX to establish the LINK SNAP file SNAP.6.

In order to use the procedure file *LOAD6, the modules listed in Figure 50 must be available on disc.

Most of the software modules listed in Figure 50 are delivered with the current version of the RET-6/VM Operating System and should already be present on the disc from the generation phase. The following modules are the exceptions.

LINK and MLLDR Command Files

#LINK	#LINDX	#RT6GN	#SWTCH	#ED1K6	#MACRO	#MLLD6	#SGMTR
#SXREF	#MERGE	#OLDRE	#SCOM	#FC6	#TF	#FORMC	#FORMT
#DS	#CF	#DT	#INIT	#MTRP	#PLRP	#SR	#AUTOR

Relocatable Software Modules

%RT6GN	%SSTCH	%EDITA	%EDITB	%LUPRN	%DRREL	%DRRPL	%INDXR
%MACRO	%MACRO	%MACR1	%MACR2	%MACR3	%MACR4	%MACR5	%MACR6
%MACR7	%CMD	%GENIX	%HELP	%TVVER	%MLLDR	%MLLDA	%MLLDB
%SGMTR	%MERGE	%ATRAN	%SCOM	%FCO	%FC1	%FC2	%FC3
%FC4	%FC5	%FC6	%TF	%FORMC	%FORMT	%AUTOR	%4AUTR

Libraries

$FMP6	$LDRLN	$PLIBN	$R6GNL	$DTCLB	$DSCLB	$ED1K6	$RBLIB
$FCL1	$FCL2	$FCM6	$TFLIB	%TVLIB	%M.LIB		

Procedure Files

(DS

INDXR Files

@CF	@DT	@INIT	@MTRP	@PLRP	@SR

Figure 50. Modules necessary to execute procedure file *LOAD6.

The procedure file (DS and all the INDXR files listed in Figure 50 are created by TRAMCON software maintenance personnel. Refer to Section 8.2.3 to see how the INDXR files are created. The procedure file (DS is used to load each of the programs in the Distributed System (DS) package. A listing of (DS can be found in Appendix C.

All the LINK and MLLDR Command files, except those for the segmented TRAMCON programs, are listed in Appendix C. Refer to Section 8.2.3 for instructions on how to produce MLLDR files for the TRAMCON segmented programs.

The modules needed to load the Pascal and Fortran language compilers were delivered on a separate tape and not included on the operating system tape.

NOTE

The power failure recovery program AUTOR was modified to fit the specific needs of the TRAMCON system. To be consistent, the new version of the program was written in Pascal. Attempts to load the Pascal version of AUTOR at generation time resulted in the generation error

GEN ERR 00

Therefore, the system developers decided that at generation time, they would load the version of AUTOR which is delivered with the other system software (module %4AUTR). After the switch is made to the new system, AUTOR would be reloaded using the TRAMCON version (module %AUTOR) and replacing the permanent program AUTOR.

10.3 Operating System Modifications

In general, it is NEVER a good idea to modify any of the operating system or other vendor-supplied software for several reasons. First, it is difficult to produce proper documentation with enough detail so that the change may be incorporated into future upgrades. Second, changing vendor-supplied code requires that the very expensive SOURCE code be purchased. Third, the changes may not operate properly from upgrade to upgrade because the vendor is free to make modifications completely independent of any user modifications. Last, but most important, the vendor can withdraw any technical support if any modifications are made by the user since the company cannot guarantee the results of those changes.

With this in mind, the changes that were done were only those absolutely necessary or resulted in a large gain for a small effort.

The TRAMCON On-Line software was designed to handle the operator interaction and eliminate any direct interaction between the operator and the operating system. During normal TRAMCON operation, very little, if any, direct operating system commands are necessary. Also, TRAMCON software does not run in the Session Monitor mode (Session Monitor is the HP operator-operating system interface that supports multiple users and includes an accounting system). The TRAMCON software is essentially the only user of the system and all terminal devices are supported through the TRAMCON software.

But the TRAMCON software also makes use of the interprocessor communications software package called DS (Distributed Systems) to perform data file transfer and to sychronize the clocks between masters. This DS software, in turn, makes use of the Session Monitor software to log on and log off at the remote end of any DS communication. Logging ON and logging OFF of a Session Monitor session causes messages to be displayed both at the session terminal device and at the System Console. Since TRAMCON is using these terminal devices to display TRAMCON information, these messages are undesirable.

To eliminate these unwanted accounting messages, the operating system modules &LOGON, &LGOFF, and &LSUB2 were modified as shown in Figure 51.

As the reader can see from Figure 51, each line that is modified is marked with the comment !TRAMCON and most of the modifications simply COMMENT out a call to the routine MESSP that would display an unwanted message on the terminal devices. The one modification to &LSUB2 changes a JMP TELL instruction to a JMP NEXT instruction, thus eliminating the message "**FMGxx REMOVED**".

The TRAMCON operator is allowed to log ON and OFF Session Monitor from a non-system console terminal using the TRAMCON SM command, which is password protected. These changes to "**logon**" and "**lgoff**" eliminate the usual messages from this use of Session Monitor also.

The software version used for these modifications is RTE-6/VM REV 2301. The line numbers specified in Figure 51 may NOT be the same for later revisions.

Changes to Module &LOGON

```
 1446 C      CALL MESSP(10001B,2H  ,-2)                 !TRAMCON
 1447 C      CALL MESSP(1,ONMS1,-34)                    !TRAMCON
 1457 C      CALL MESSP(2,ONMS1,-24+ITMP1)              !TRAMCON
 1467 C      CALL MESSP(10001B,ONMS3,-58)               !TRAMCON
```

Changes to Module &LGOFF

```
   35        DIMENSION OFPRG(8),PGRM(3),DSEC(2),DCPU(2)  !TRAMCON (6) := (8)
   70        DATA OFPRG/2HOF,2H ,,3*2H  ,2H,8,2H,N,2HP / !TRAMCON
  573        CALL MESSS(OFPRG,15)                        !TRAMCON 12 := 15
  885 C      CALL MESSP(10001B,DMES,DMLEN)               !TRAMCON
 1099 C      CALL MESSP(10001B,2H  ,-2)                  !TRAMCON
 1100 C      CALL MESSP(1,OFMS1,-34)                     !TRAMCON
 1106 C      CALL MESSP(2,OFMS1,-24+NAML)                !TRAMCON
 1213 C      CALL MESSP(10001B,OFMS3,-54)                !TRAMCON
 1226 C      CALL MESSP(10001B,OFMS4,-64)                !TRAMCON
 1234 C      CALL MESSP(10001B,OFMS5,-54)                !TRAMCON
 1239        IF(IAND(MAIL,100000B).EQ.0) GOTO 450        !TRAMCON
 1240 C      CALL MESSP(10001B,2H  ,-2)                  !TRAMCON
 1245 C5500 CALL MESSP(10001B,14HEND OF SESSION,-14)     !TRAMCON
```

Changes to Module &LSUB2

```
  300        JMP NEXT                                    !TRAMCON was JMP TELL
```

Figure 51. Modifications to &LOGON, &LGOFF, and &LSUB2.

11. DATA STRUCTURE

This section describes the Pascal CONST and TYPE definitions that describe the data (memory-resident and disc-resident) used by the TRAMCON software. All user defined constant and type definitions are collected in one software module named [RECR3. This central location makes it easier for the software maintainer to find definitions and trace the use of data within the software modules.

The vast majority of the data, including all the information that describes the current state of the communications system, is placed into a memory area called EMA so that it can be shared by several programs. This data is accessed through one global address called "**heap**".

Other identifiers that are used by several programs are grouped together into a software module called [TRVAR which can be included by any program module. This avoids having to explicitly declare these variables in each program module. Also, if any of these variables are changed, the code in [TRVAR needs to be changed once, and all the programs using the variables need only be recompiled and re-linked.

11.1 Type and Constant Definitions [RECR3

The file [RECR3 contains CONST and TYPE definitions shared by more than one TRAMCON On-Line program. The following are short descriptions of each identifier defined in [RECR3. The first portion of [RECR3 is a CONST section contain constant definitions that, when used, make the code more readable and reduces the amount of redundant constants generated by the compiler throughout the software. Following the CONST section is a lengthy TYPE section that contains several basic TYPE definitions followed by the TYPE definitions that describe both the static (Configuration data base) data and the dynamic (created and changed at run-time) data that are stored in the HEAP. All the components of the HEAP are described first, followed at the very end by the definition of the HEAP itself (identifier "**heap**"). The organization of this module is also based on the general guidelines "**define a TYPE or CONST immediately before its first reference**" and "**group similar TYPES or CONSTANTS, such as PACKED ARRAY OF CHAR, together**". Many CONSTANTS are defined here so that they can be more easily changed. For example, to change the power failure interrupt LU "**pflu**" from 14 to 33, the only source code change required is to the CONSTANT definition in this module. All the software references to this CONSTANT still refer to the unchanged identifier "**pflu**". Of course, the modules that reference the power failure LU must be recompiled and reloaded.

11.1.1 CONST Section of [RECR3

```
one_minute = -6000;
no_abort_bit = -32768; {bit 15}
no_wait = -32768; {bit 15}
clone_bit = 2048; {bit 11}
```

The first four constants, shown above, are used in various calls to the routine EXEC. The value "**one_minute**" is the optional parameter for a Control (3) EXEC Request used to set the amount of time that RTE will wait for a remote unit response before it sends a time-out interrupt to the segment I/O drivers DVA76 or DVA77. The value is the negative number of tens-of-milliseconds to wait.

NOTE

Negatave numbers are used often for counting purposes on the HP-1000 because the instructions set includes INCREMENT and TEST instrucions, but doesn't include a DECREMENT instruction.

The other three constants, "**no_abort_bit**", "**no_wait**", and "**clone_bit**" are used to modify the program schedule as described in the Programer's Reference Manual, pp. 2-10, and 2-58.

```
     audible_lu = 14;  {audible alarm lu}
     pflu = 13;        {Power Fail lu}
```

The identifiers "**audible_lu**" and "**pflu**" are defined to equal the Logical Unit numbers that are associated with the audible alarm panel (Relay Output Interface in I/O Slot 22 Octal) and the power failure interrupt vector address (4), respectively.

```
     stack_alloc_size = 50;  EMA_max = 190000; {190-page EMA partition #11}
```

The constant "**stack_alloc_size**" is used by the routine "**allocate_EMA**" to allocate stack space to any program requesting it. This constant specifies the unit amount of stack space (currently 50 words) that will be allocated. Any program requesting stack space must specify the number of 50-word portions it wants. The only programs currently requesting stack space are MTRP and PLRP. The next constant, "**EMA_max**", is used by the routine "**allocate_EMA**" to define the size of the HEAP for each program that uses the shareable HEAP. The current size is set to 190,000 words and should be expanded to 225,000 words as recommended in Section 3.2 of this manual. This value matches the 190-page shareable EMA memory partition called SHAR1, which is set up at system generation time and can be modified without regenerating, using the reconfiguration bootup procedure (see Section 10). If the shareable EMA partition is changed, this constant must be changed to a corresponding value and all TRAMCON On-Line software recompiled and reloaded.

```
     max_pf_rec = 410;
```

The constant "**max_pf_rec**" specifies the maximum number of power fail messages that will be stored in the power failure disc file (PF. This constant is used by the program PF as a LOOP terminator when displaying the power failure messages.

```
     nocctl_shared = 'NOCCTL,SHARED';  shared = 'SHARED';
```

The string constants "**nocctl_shared**" and "**shared**" are used by various programs when they open files. Both strings specify the shared option, which allows a file to be open to more than one program at the same time. The "**nocctl_shared**" constant is used whenever a program opens the global (declared in INCLUDE module [TRVAR) text file "**outunit**" for output to a terminal display.

```
    date_file_name = '(DATE::2';       {Ref by HR,CO,SETCL,SETDT,SR,DT,
                                        AUTOR,X,CMMD,BROADC,TIMPAS,TIMSET}
    arch_file_name = '(ARCH:TR:10';    {Ref by MTRP ,PLRP ,AL ,INIT , HR }
    statz_file_name = '(STATZ:TR:10';  {Ref by US , INIT , HR }
```

The file name definitions above allow the files to be relocated, renamed, or given a different security code without changing the source code anywhere but in the above constant definitions.

```
    nill = -1;  null = #0;  site_category = nill;
```

The constant -1, given the name "**nill**" above, is one of the most widely used constants in the TRAMCON On-Line software. A particular use of the -1 constant is given yet another name above. The identifier "**site_category**" defined above refers to the index value for the site alarm/status information in the "**linkend**" array. The identifier "**null**" is a less widely used name for the unprintable ASCII character code with numerical value 0.

```
    nbr_bins = 16; {number of histogram bins}
```

The constant "**nbr_bins**" is a critical factor in determining the size of the largest data file, (HIST, and the amount of EMA allocated for the current hour's parameter readings. The value "**nbr_bins**" has only one reference that affects the size of EMA data storage and the size of the disc files (HIST, (PHIST and (CURVE. That reference is below in the definition of the TYPE "**hist_array**" which, in turn, is used below in the definition of the EMA data record "**current_link_status_record**" and the record definition "**parm_record**". The record "**parm_record**" is the definition of records in the disc file (CURVE. The TYPE "**hist_array**" is used as the definition of the records in the disc file (HIST. The constant "**nbr_bins**" is also referenced by the programs MTRP and PLRP through routine "**process_response**", HR, PH, and CC. These programs use "**nbr_bins**" as a loop terminator.

```
    ss_alfa = 'SS'; map_alfa = 'MA'; diag_alfa = 'dI';
    al_alfa = 'AL'; pa_alfa = 'PA'; pc_alfa = 'PC'; cn_alfa = 'CN';
    archive_alfa = 'AR'; scenario_alfa = 'SC'; msg_alfa = 'MSG   ';
```

The two-letter constants above are used primarily by the routine "**update_displays**" in $MPLIB to determine which display is currently painted on a given terminal screen so that the proper display update routine can be called to refresh the screen. These constants are also referenced by the routines "**sched_dsp_prog**" and "**update_displays**" in program CMMD.

```
    max_chars_per_response = 270;
```

The constant "**max_chars_per_response**" is referenced by the remote unit response handling routines "**pm_Initialize**" and "**get_answer**" as the input buffer length (in bytes) for accepting a response from a remote unit. Routine "**get_answer**" also uses this constant as an upper bound on the length of a remote unit response and sets the value of global VAR "**res_len_ok**" to false if the response is longer than "**max_chars_per_response**".

max_chars_per_cmd = **80**; {maximum length of TRAMCON command}

The constant "**max_chars_per_cmd**" is used in the definition of the operator keyboard input buffer TYPE "**cmd_str**" defined below. Program KYBRD issues keyboard read requests using a string variable of TYPE "**cmd_str**" as the input buffer and the CONST "**max_chars_per_cmd**" as the input buffer limit.

a2d_card_select = **#112; a2d_nbr_values** = **#15**; {DATALOK10 remote unit}

The A/D Mux card on the DATALOK10 Model 1E remote unit is capable of sensing and reporting 16 analog values. The card can be programed to report from 1 to 16 values each time the remote unit is polled. The above two CONSTANTS, "**a2d_card_select**" and "**a2d_nbr_values**", are sent to the remote unit by the program POLL as bytes 4 and 5 of the POLL message to program the A/D Mux card to report all 16 values.

```
crt_msg_len = 67;     {crt_msg_record length in chars}
max_crt_msg = 5;      {max msgs buffered per crt}
```

The constant "**crt_msg_len**" is the size (in words) of the record "**crt_msg_record**" defined below and is used by the program MSG as the buffer length limit for the CLASS GET statement, which reads in a message to be added to the message array for the given terminal. The constant "**max_crt_msg**" is used in the TYPE definition "**crt_msg_ordinal**" below which, in turn, is used to size the arrays "**msg_priorities**", "**msg_lengths**" and "**msgs**" in the HEAP record "**heap^.current_crt[crtord]**".

line_of_sight='M';satellite = 'S';troposcatter = 'T'; fiber_optics='F';

The four constants above equate some very informative identifiers with the first letters used in the link identifiers that are found in the Configuration data record "**linkend_record**" defined below.

latching = **1**;   **momentary** = **0**; {remote unit relay types}

The identifiers "**latching**" and "**momentary**" are equated to the numeric values 1 and 0. These identifiers are referenced in the program SW.

inactive = **0**; **monitor** = **1**; **poller** = **2**; {segment status values}

The segment status values 0, 1, and 2 are given the names "**inactive**",

"monitor" and **"poller"**, respectively. These identifiers are referenced in the program PM which processes the operator request to alter the current status of a given segment. The INACTIVE status setting was never implemented.

```
    {The following are names of keyboard FUNCTION keys (SOFTKEYS)}
    soft1 = '1'; soft2 = '2'; soft3 = '3'; soft4 = '4'; soft5 = '5';
    soft6 = '6'; soft7 = '7'; soft8 = '8';
```

On the top row of every keyboard are eight function keys labeled **"f1"** through **"f8"**. These function keys are also referred to as SOFT keys. They are soft keys because they can be programed, in a sense, to convert a single physical key press into a set of key presses. That is, each soft key has a string of characters associated with it that can be defined by the user. Each time a particular soft key is pressed, the terminal firmware transmits the associated string of characters to the computer rather than the one-key code that uniquely identifies the soft key itself. In the TRAMCON on-line software, there has been a trend to convert as much of the operator key strokes to single soft key presses as possible, thus reducing the burden on the operator to accomplish the TRAMCON functions.

In the TRAMCON software, each of the soft keys is defined to send two characters to the computer when pressed. The first character generated by each soft key is the ASCII decimal digit corresponding to the soft key number. The second character is the ASCII RETURN (12) character. For instance, soft key **"f1"** causes the terminal to send the two ASCII characters **"1"** and **"<RETURN>"** to the terminal interface driver. Because the input request is a NORMAL READ request, the driver (DVX05) interprets the <RETURN> key as input termination and does not place the <RETURN> character into the input buffer. Therefore, the program issuing the read request sees only the first character of the soft key definition. The characters generated by the soft keys, namely **"1"** through **"8"**, have been given the much more readable names of **"soft1"** through **"soft8"** in the above constant definitions.

```
    {The following are CRT types supported by TRAMCON}
    HP-2647F = 0; HP-2627A = 1; HP-2397A = 2; HP-2623 = 3; HP-2393A = 4;
```

The four constant definitions above allow the programmer to refer to terminal types using the model numbers such as **"HP-2647F"** rather than the corresponding decimal number, such as 0, that is found in the Configuration data base **"crt_record"** and is reset by the program LO each time a user logs-on to that device. The numeric value of the terminal type constants is currently stored in the Configuration data base record **"crt_record"** field **"terminal_type"**. Since this value is recalculated each time someone logs-on at a terminal, this value could be moved to the HEAP record **"current_crt"** and removed from the Configurator data entry.

```
    {The following are basic colors for HP-2627A and HP-2397A CRTs }
    red='1'; green='2'; yellow = '3'; blue = '4'; magenta ='5'; cyan = '6';
```

{Alphanumeric color pairs for SS display }
red_on_blue = '5'; blue_on_yellow = '6'; blue_on_red = '7';

The constant definitions above give descriptive color names to the nondescript ASCII decimal digits. These identifiers are used in output instructions to the terminal displays to define the colors to be used for a particular display. Color was a late addition to the TRAMCON On-Line system and is not well developed. Basically, "**green**", "**yellow**", **and** "**red**" have been used to color the text for the status conditions OK (green), minor (yellow), and MAJOR (red), respectively. All the colors listed above are used to display text information. The second row of color constants is used to display text of the first color mentioned against a background of the second color. For example, the "**red_on_blue**" constant is used to display the heading lines for all displays in red letters over a blue background. The "**blue_on_yellow**" and "**blue_on"red**" constants are used by the SS program for minor and major conditions respectively.

HP-2631G = 1; HP-2932A = 2; HP-2934A = 3;

The constant definitions above give meaningful names to the printer types supported by the TRAMCON software and are referenced by program LO each time a new user logs-on to the TRAMCON system. The HP-2631G printer uses an IEEE-488 interface and connects to the older HP-2647F terminal. Both unit types are being replaced. The other two printer types are nearly identical and use an RS232 serial interface to connect to an HP-2627A or an HP-2397A terminal. The numeric value of the printer type constants is currently stored in the Configuration data base record "**crt_record**" field "**printer_type**".

MAJOR = 2; minor = 1; status = 0;

The names "**MAJOR**", "**minor**", **and** "**status**" are associated with the two-state data type in the Configuration data base record "**equipment_record**" defined below. The two-state type indicator "**alarm_type**" is found in the definition of the record "**two_state_record**", which is, in turn, part of the "**equipment_record**".

The next set of constant definitions are various critical dimensions of the Configuration data base. By using CONST definitions, the data base sizing can be altered by changing only the desired CONST and recompiling and reloading. Since the identifier associated with the changed constant value did not change, no further changes are necessary to the source code. The following constant values were chosen based on the DRAMA radio equipment requirements plus an expansion cushion of 10 to 15 percent.

max_archive_record = 200; {number of archive records per-remote-unit}

The constant "**max_archive_record**" defines how many records in the disc file (ARCH are assigned to each remote unit defined on the given master. Currently, 200 records are reserved for each remote unit. Therefore, 200 separate events can be archived on disc for any given remote unit. To keep more information for each remote unit, simply increase this constant, then recompile and reload $MPLIB, MTRP, PLRP, and AL.

max_segments_per_net = 25;

The constant "**max_segments_per_net**" is used in the definition of the array "**segment_info**" in the Configuration data base record "**network_record**" below. It is also used by the program DT as a loop·terminator when searching through the "**segment_info**" array in the "**network_record**".

max_masters_per_net = 30;

The constant "**max_masters_per_net**" is similar to "**max_segments_per_master**" except it is used in the "**master_info**" array of the "**network_record**".

max_links_per_net = 250; **max_sites_per_net** = 250;

The constant "**max_links_per_net**" is used below in the TYPE definition "**link_def_ptr**", which, in turn, sizes the Configuration data base array "**links_record**" thus affecting the record definition for the disc file (LINKS. A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (LINKS file with the new record size. Refer to Section 11.4 of this manual for a discussion about changing record sizes in the disc files.

max_segments_per_master = 4;

The constant "**max_segments_per_master**" is used below to calculate CONST "**max_sites_per_master**", to define an upper bound to the TYPE "**master_segment_ordinal**", and to dimension the array "**segment**" in the Configuration data base record "**master_record**". The range TYPE "**master_segment_ordinal**" affects the size of the "**date_record**", which is used in the disc file (DATE and the dimension of the HEAP arrays "**time_it**" and "**time_val**".

Even though the present TRAMCON design supports only two segments, this value was left at four so we would not change the size of the master record, which would require us to redo both the On-Line and the Configuration software. More importantly, this would require a change in the record sizes for the disc files (MAST and (DATE. Changing the size of records in the fixed record length disc files is a major job. The gain here would just be a few words of space in the files (MAST and (DATE and in the HEAP, which stores the master record during TRAMCON operation. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_crts_per_master = 5;

The constant "**max_crts_per_master**" is used below to set an upper bound for the TYPE "**master_crt_ordinal**", which, in turn, is used to dimension the array "**crt_ptr**" in the Configuration data base record "**master_record**" and the array "**current_crt**" in the main HEAP record "**heap_ptrs**" below.

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (MAST file with the new record size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_remotes_per_segment = 21;

The constant "**max_remotes_per_segment**" is one of the most important constants in the TRAMCON software. This constant is used to set an upper bound for the TYPE "**segment_remote_ordinal**", which, in turn, is used to dimension the array "**remote_info**" in the Configuration data base record "**segment_record**" and to dimension the array "**remote_status**" in the HEAP record "**segment_status_record**". The current value of 21 was chosen because, by using one line for each remote unit and allowing for three lines of heading and time-stamp, the information for up to 21 remote units could be displayed on a single 24-line display. A recent change was made to the format of the SS display to actually accommodate a 21-remote-unit segment.

This constant is a limit on the number of PHYSICAL remote units that can be defined for any segment regardless of how many of those PHYSICAL units are grouped together into multiple (LOGICAL) units. For example, if a hypothetical segment has two LOGICAL remote units defined, each of which consisted of 10 PHYSICAL remote units, only one more remote unit could be defined on that segment.

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (SEG file with the new "**segment_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_trunks_per_segment = 100;

The constant "**max_trunks_per_segment**" is used as an upper index limit in the ARRAY TYPE "**pcm_histogram_array**", which, in turn, is used as the TYPE of the field "**pcm_counts**" in the HEAP record "**segment_status_record**" and in the definition of the record "**pcm_histogram_record**". This record definition is used for the disc file (PHIST. This constant is used as an upper limit for the index of the array "**trunk_info**" in the Configuration data base record "**segment_record**". Constant "**max_trunks_per_segment**" is also used by programs PC and PH as a LOOP terminator.

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (SEG file with the new "**segment_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_masters_per_segment = 4;

The constant "**max_masters_per_segment**" is used as an upper bound for the index of the ARRAY TYPE "**alt_mast_array**", which, in turn, is the TYPE of the field "**alternate_masters**" in the Configuration data base record "**master_record**" described below. This constant limits the number of masters that can have any particular segment defined in their data base.

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (MAST file with the new "**master_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

    max_linkends_per_remote = 3;

The constant "**max_linkends_per_remote**" is another widely used constant. This constant determines how many sets of communication equipment can be monitored by one PHYSICAL remote unit. This constant is used to set an upper bound on the array "**linkend_info**" in Configuration data base record "**remote_record**". Constant "**max_linkends_per_remote**" is used as an upper bound on the range TYPE "**category_ordinal**", which, in turn, is used to dimension the array TYPE "**category_array**", the arrays "**alarms**", "**a2ds**", and "**digitals**" in record TYPE "**unpacked_response_record**" and the array "**archive_alarms**" in the record TYPE "**archive_alarm_status_record**". Array "**archive_alarms**", in turn, affects the size of the record "**archive_record**" used in disc file (ARCH. This constant also affects the dimension of the arrays "**cal_curves**", "**a2d_bottom**", "**a2d_top**", "**a2d_amber**", "**a2d_red**", "**digital_bottom**", "**digital_top**", "**digital_amber**", and "**digital_red**" in record TYPE "**parm_record**", which is part of the HEAP "**segment_status**" data. It affects the dimension of TYPE "**cn_record**", which is both part of the HEAP (**heap^.segment_status[segord].remote_status[remoteord].counts**) and the record definition for the disc file (CN. The last affect of this constant is on the HEAP array "**cat_status**", which is in record TYPE "**remote_status_record**".

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (REMOTE file with the new "remote_record" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

    max_sites_per_trunk = 18;

The constant "**max_sites_per_trunk**" limits the number of nodes, including end points, for a communications DIGROUP (also known as a TRUNK). This constant sets an upper limit on the index for the array "**nodes_in_trunk**" in the Configuration data base record "**trunk_record**" defined below.

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (TRUNK file with the new "**trunk_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

```
     max_2states_per_link = 144;  {72 normal + 12 analog threshold + 44
                                    digital threshold + 16 combination}
```

The constant "**max_2states_per_link**" limits the number of two-state alarm/status values on a set of communications equipment that can be monitored by the TRAMCON On-Line system. THIS IS A DATA BASE LIMITATION AND SAYS NOTHING ABOUT ANY PHYSICAL REMOTE UNIT LIMITS THAT CURRENTLY ARE MORE RESTRICTED. The current setting, 144, was chosen based on the DRAMA communications equipment requirements, plus a 10 to 15 percent cushion. This constant sets the upper bound for the subrange TYPE "**link_2state_ordinal**" which is defined below. Through "**link_2state_ordinal**", this constant affects: (1) the size of the HEAP record "**current_link_status_record**", (2) the run-time data base disc file records "**archive_record**" [file (ARCH] and "**cn_record**" [file (CN], and (3) the Configuration data base record "**equipment_record**".

A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (EQT file with the new "**equipment_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

```
analog_start = 72;  {Starting position of analogs within 2-state array}
digital_start = 84; {Starting position of digitals within 2-state array}
combo_start = 128;  {Starting position of combos within 2-state array}
```

From the comment shown above for "**max_2states_per_link**", notice that the total number of two-states is partitioned into types of two-states. The constants "**analog_start**", "**digital_start**", and "**combo_start**" mark the ordinal for the first two-state value of their respective types. The current settings allow for 72 normal two-state alarm and/or status points, 12 analog threshold crossing values (6 amber and 6 red), 44 digital parameter threshold crossings (22 amber and 22 red), and 16 combination alarms. These constants are used by the response processing procedures "**unpack_response**" and "**process_response**" to position the alarm data in the generic global response buffer "**unpacked_response**" and to analyze the two-state information using the same positioning scheme to locate corresponding information in the Configuration data base equipment record. If more of a particular two-state type (normal alarm/status, analog parameter, digital parameter, or combo) is desired, the appropriate constants must be changed here. The gain for the expanded type will be at the expense of some other two-state type. Also, these constants apply to ALL equipment monitored by TRAMCON. That is, no matter what communications equipment is being monitored at a given location only six analog parameter values can be monitored with the above settings. The first 72 two-state values are those actually reported by the remote unit. The other two-state types (analog, digital, and combo) are derived from the real data. The analog and digital threshold-crossing two-state indicators are determined by comparing the analog and digital parameter values reported with the thresholds set by the operator. The combo two-state values are derived from several real two-state values according to logical expressions that were specified in the equipment record by the Configurator.

NOTE

The values for "**digital_start**" and "**combo_start**" are dependent upon the two constants "**max_a2ds_per_link**" and "**max_digitals_per_link**" defined below. To further ensure consistency, the definitions for "**digital_start**" and "**combo_start**" could be placed after "**max_a2ds_per_link**" and "**max_digitals_per_link**" to read as follows:

```
digital_start = analog_start + max_a2ds_per_link * 2;
combo_start = digital_start + max_digitals_per_link * 2;
```

```
max_a2ds_per_link = 6;
max_digitals_per_link = 22;
```

The two constants "**max_a2ds_per_link**" and "**max_digitals_per_link**" place an upper bound on the number of analog and the number of digital (pulse count) parameters that can be monitored by the TRAMCON system for each category (site or linkend). THESE ARE DATA BASE LIMITATIONS AND SAY NOTHING ABOUT ANY PHYSICAL REMOTE UNIT LIMITS THAT CURRENTLY ARE MORE RESTRICTIVE. These values are used in the calculation of "**max_histos_per_link**" below and as an upper bound on the range TYPEs "**a2d_ordinal**" and "**digital_ordinal**", which are defined below. These two range types, in turn, influence the size of the global record "**unpacked_response_record**", the Configuration data base record "**equipment_record**" and the HEAP records "**current_link_status_record**" and "**remote_status_record**".

A change to either of these two constants would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (EQT file with the new "equipment_record" size. Refer to Section 11.4 of this manual for a discussion about changing record sizes in the disc files. As you can see, increasing these values by even a small amount will significantly increase the storage requirements for the HEAP (EMA). Increasing either of the above values by one requires approximately 1200 words of HEAP storage.

```
max_histos_per_link = max_a2ds_per_link + max_digitals_per_link;
```

The constant "max_histos_per_link" is NOT used in any of the data TYPE definitions below, but is referenced in the program HI, approximately line 86 and the program HR, approximately line 69. With such limited use, this constant could be removed in some future low priority housecleaning of [RECR3.

```
max_relays_per_link = 20;
```

The constant "max_relays_per_link" is used to set an upper bound for both the subrange TYPE "relay_ordinal" and the index for the array "relays" in the Configuration data base record "equipment_record" below. A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (EQT file with the new "equipment_record" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

> NOTE
>
> To further ensure consistency, the subrange TYPE "**relay_ordinal**" should be substituted for "**0..max_relays_per_link-1**" in the definition of array "**relays**" in the Equipment record below.

max_combos_per_link = 16;

The constant "**max_combos_per_link**" is used to set an upper bound for the index for the array "**combos**" in the Configuration data base record "**equipment_record**" below. A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (EQT file with the new "**equipment_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_specific_names = 40;

The constant "**max_specific_names**" is used to set an upper bound for the index for the array "**specific_name**" in the Configuration data base record "**link-end_record**". This value is the maximum number of two-state indicators that can be uniquely defined for any given link end. A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (LINK file with the new "**link-end_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_counts_per_link = 20;

The constant "**max_counts_per_link**" specifies how many two-state values, of the possible "**max_2states_per_link**", can be designated to be counted for any given link end. It is used to set an upper bound for the index for the array "**counts_array**", which is defined below. The array "**counts_array**", in turn, affects the size of the record "**cn_record**". "**cn_record**" is used as the record description for disc file (CN and in the HEAP as part of the record "**remote_status_record**", each of which is described below.

dictionary_size = 14000;

The constant "**dictionary_size**" defines the size, in bytes, of the Configuration data base dictionary, which is described below. The previous value of 4000 used in Version 1.7.3 was found to be inadequate for an average field system dictionary. The new value of 14000 should be more than sufficient for any future needs and should NOT have to be changed. The constant "dictionary_size" is used to set an upper bound for both the pseudo pointer TYPE "dictionary_ptr" and the index for the character array "dictionary" both of which are defined below.

expression_size = **16**;

The constant "**expression_size**" limits the number of nodes (i.e., LOGICAL operators or operands) that can be used to compose a combination two-state alarm. It is used to set an upper limit for the subrange TYPE "**expression_ordinal**", which is used as the index for the TYPE "**expression_tree**", which, in turn, is used to define the field "**expression**" in the record "**combo_record**". The record "**combo_record**" is a field in the Configuration data base record "**equipment_record**". All of these TYPES are defined below. A change to this constant would require that the Configurator program CONFI be recompiled and reloaded so that it can be used to create a new (EQT file with the new "**equipment_record**" size. Refer to Section 11.4 for a discussion about changing record sizes in the disc files.

max_words = **6**;

The constant "**max_words**" specifies how many "**dictionary_word**"s can be used to construct the English phrase TYPE "**name_list**" described below. It is used to set an upper limit for the index into the array TYPE "**name_list**". Programs that display such things as alarm descriptions or parameter names use this constant as a LOOP terminator.

dictionary_word_size = **30**;

The constant "**dictionary_word_size**" places an upper limit (number of characters) that can constitute a WORD in the Configuration data base dictionary. This constant is used as the upper limit for the index into the character array type "**dictionary_word**" defined below. This constant is used by the routine "**read_dict**" (approximately lines 442, 446 and 447) in $TRLIB as a loop terminator.

max_links_per_segment = **25**;
max_remotes_per_master=max_remotes_per_segment*max_segments_per_master;
max_equipments_per_master = **20**;

NOTE

The constants "**max_links_per_segment**", "**max_remotes_per_master**", and "**max_equipments_per_master**" have NO current references. These constants are used by the Configurator and were anticipated to be used by the on-line software.

```
bell = #7;   esc = #27;     colon = ':';   slash = #47;     space_bar = ' ';
ret = #13;   delete = #127;  lf = #10;     form_feed = #12;  tab = #9;
```

The single character constants defined above serve to give convenient and readable identifiers to commonly used ASCII characters. All of these constant definitions, except "**colon**" and "**space_bar**", give a very readable name to unprintable ASCII CONTROL characters (refer to the Pascal/1000 reference Manual, p. 5-30). Most of these constants are used in terminal output statements. Since much of the output is buffered (see Section 9), the two most commonly used constants from above are "**ret**" and "**lf**" because they must be manually specified at the end of logical screen lines. The terminal driver supplies these two characters automatically only at the end of a physical WRITE, such as WRITELN. The next most commonly used constant above is "**esc**" because much of the output to the screen is screen control and setup information, which is triggered by an escape character. These constants are referenced by the display program scheduler CMMD, every display program, and the display refresh routines in $MPLIB.

cmd_alfas1 = 'UNMASSALARPAMEHEHICNPCPHSWCRCCCFPOACINENDTPM';
cmd_alfas2='OPSESMSIDEPRLSSCSRMSOLCOSTDILOWHLUEQUPDNOFRUVEUSPFPWEDIL';

The last two entries in the CONST section of [RECR3 are the two string constants "**cmd_alfas1**" and "**cmd_alfas2**". These strings are used by the TRAMCON command parsing routine "**parse_it**" in program CMMD. The use of these string constants and the command parsing processing is detailed in Section 5.2. Basically, each TRAMCON command is uniquely identified by a two-character command code that must be the first item in any command entry. Each pair of characters, starting at the beginning of "**cmd_alfas1**" and continuing to the end of "**cmd_alfas2**" is one of these two character command identifiers. For command parsing convenience, the real command codes are bracketed by the command codes "**UN**" which stands for undefined and "IL" which stands for illegal. There is a one-to-one correspondence between the character pairs in these strings and the elements in the class called "**cmds**", which is defined below. For example, the first two characters in "**cmd_alfas1**", "**UN**" corresponds to the first element in "**cmds**", "**un**". The two strings can be considered as one long string and were broken into two because the long string literal would not fit on an 80-character line for easy reading.

11.1.2 TYPE Section of [RECR3

Though it is not the only one, common use is the prevailing reason for placing a TYPE definition in the module [RECR3. Another good reason has to do with the strict typing rules imposed by the Pascal language. All procedures and functions must be predefined, and the type and number of the formal parameters must exactly match the type and number of the actual parameters in each call of the given procedure or function. By placing the type definitions here, both the formal procedure declarations and the VAR declarations can make use of them. Another good reason for placing type definitions here is to give readable names to otherwise vague numeric range types.

All of the TYPES describing the records in the Configuration data base files and those TYPES describing the organization of the shared memory, called the HEAP, are contained in this section of module [RECR3.

byte = 0..255; nibble = 0..15; INT = -32768..32767;

The three range types above can be considered as additions to Pascal's predefined BASIC types. By far the most popular of the three is "**INT**", which is the one-word (16-bit) version of the predefined the two-word (32-bit) type INTEGER. Most integer values in the TRAMCON software are one-word "**INT**" values because they take half the space and the arithmetic is twice as fast. The value "**byte**" defines an 8-bit integer and is used in a few places, especially in packed structures. The value "**nibble**" defines a 4-bit integer but is rarely used in packed structures.

```
cmds = (un,ma,ss,al,ar,pa,me,he,hi,cn,pc,ph,sw,cr,cc,cf,po,ac,
        ih,en,dt,pm,op,se,sm,si,de,pr,ls,sc,sr,ms,ol,co,st,
        di,lo,wh,lu,eq,up,dn,off,ru,ve,us,pf,pw,ed,il);
```

The class "**cmds**" enumerated above is the list of official TRAMCON commands and is referenced by the command parsing routine "**parse_it**" in the program CMMD. Notice the one-to-one correspondence between the elements of "**cmds**" and each two-letter pair in the string constants "**cmd_alfas1**" and "**cmd_alfas2**" above. As explained in Section 5.2, adding or deleting commands requires that an element be added to or deleted from this class and the corresponding two letters be added to or deleted from one of the string constants, "**cmd_alfas1**" or "**cmd_alfas2**". It is very important that the order be maintained. First, the bracketing fictitious commands "**un**" and "**il**" must never be deleted. Second, all true commands must be placed between these two commands. Last, there is another program that references the class "**cmds**". Program US gathers statistics on the use of the TRAMCON commands. Currently, US monitors the use of the commands between the commands "**un**" and "**us**". These statistics are stored in the one record on the disc file (STATZ. If any command is added or deleted between the commands "**un**" and "**us**", the record size for file (STATZ would change. This new record must be written to file (STATZ before the TRAMCON system will bootup properly. The problem caused by changing the record size for a type 2 (fixed-length record) disc file is explained in detail in Section 11.4.

```
two_chars        = PACKED ARRAY[1..2] OF CHAR;
three_chars      = PACKED ARRAY[1..3] OF CHAR;
four_chars       = PACKED ARRAY[1..4] OF CHAR;
five_chars       = PACKED ARRAY[1..5] OF CHAR;
six_chars        = PACKED ARRAY[1..6] OF CHAR;
seven_chars      = PACKED ARRAY[1..7] OF CHAR;
eight_chars      = PACKED ARRAY[1..8] OF CHAR;
ten_chars        = PACKED ARRAY[1..10] OF CHAR;
time_str         = PACKED ARRAY[1..12] OF CHAR;
fourteen_chars   = PACKED ARRAY[1..14] OF CHAR;
sixteen_chars    = PACKED ARRAY[1..16] OF CHAR;
twenty_chars     = PACKED ARRAY[1..20] OF CHAR;
twenty8_chars    = PACKED ARRAY[1..28] OF CHAR;
forty_chars      = PACKED ARRAY[1..40] OF CHAR;
sixty_chars      = PACKED ARRAY[1..60] OF CHAR;
cmd_str          = PACKED ARRAY[1..max_chars_per_cmd] OF CHAR;
response_str     = PACKED ARRAY[1..max_chars_per_response] OF CHAR;
soft_key_labels_type = PACKED ARRAY[3..8] OF twenty_chars;
time_string = STRING[28];
```

Most character strings used in the TRAMCON On-Line software are defined above and are PACKED ARRAYS OF CHAR rather than the newer HP extended type STRING because the type STRING has a header part associated with the actual string body. This header is difficult to deal with when passing strings as parameters, especially when using predefined routines from the relocatable library that are tailored to FORTRAN 77.

```
four_bits  = 0..15;  five_bits  = 0..31;  six_bits  = 0..63;
seven_bits = 0..127; eight_bits = 0..255; nine_bits = 0..511;
```

The range types above are sub-machine-word size and are very useful for defining fields in PACKED RECORD definitions below. The names are self-explanatory and represent the minimum amount of bits needed to represent every integer in the subrange. The subranges do not include negative numbers because the primary intent of these types is to cause the allocation of a fixed number of bits to a field and to allow the software to reference this fixed number of bits.

```
crt_msg_ordinal         = 1..max_crt_msg{5};
master_segment_ordinal  = 0..max_segments_per_master{4} - 1;
category_ordinal        = site_category..max_linkends_per_remote{3}-1;
relay_ordinal           = 0..max_relays_per_link{20} - 1;
link_2state_ordinal     = 0..max_2states_per_link{144} - 1;
a2d_ordinal             = 0..max_a2ds_per_link{6} - 1;
digital_ordinal         = 0..max_digitals_per_link{22} - 1;
master_crt_ordinal      = 0..max_crts_per_master{5} - 1;
segment_remote_ordinal  = 0..max_remotes_per_segment{21} - 1;
expression_ordinal      = 0..expression_size{16} - 1;
```

The set of subrange types above is extremely important to the software that references the Configuration data base data and the dynamic HEAP data corresponding to the Configuration data. These values are extensively used in the Configuration data base TYPE definitions and the dynamic HEAP TYPE definitions below. All of them are used as index subranges for various arrays in the data defined below and thus have the term "ordinal" as part of their names. Most of the subranges start at 0 and are bounded on the upper end by one less than some previously defined constant. This is one of the programming conventions chosen by the TRAMCON software developers and the reasons for this choice are enumerated in Section 8.4.

The subrange "**crt_msg_ordinal**" is used as the index range for the arrays "**msg_ords**", "**msg_segords**", "**msg_remoteords**", "**msg_priorities**", "**msg_lengths**", and "**msgs**" in the HEAP record "**current_crt**" all defined below. The upper bound of this subrange is based on the value of constant "**max_crt_msg**", which is defined above.

The subrange "**master_segment_ordinal**" is used as the index range for the arrays "**latlons**", "**remotes_displayed**", "**alarms_acknowledged**", "**remotes_to_print**", and "**alarms_inhibited**" in the HEAP record "**current_crt**", and for the arrays "**time_it**", "**time_val**", "**EMA_start**", "**EMA_end**", "**EMA_required**", and "**resp_stats**" in the main HEAP record "**heap_ptrs**". This subrange is also used as an index range in the arrays "**transmission**" in the HEAP record "**statz_record**" and in the arrays "**segnames**" and "**nremotes**" in the global record "**date_record**" all defined below. The upper bound of this subrange is based on the value of constant "**max_segments_per_master**", which is defined above.

The subrange "**category_ordinal**" is used as the index range for the arrays "**alarms**", "**a2ds**", and "**digitals**" in the global data record "**unpacked_response_record**", "**archive_alarms**" in the global data record "**archive_record**" and "**cat_status**" in the HEAP record "**remote_status_record**". This subrange is also used as an index range in the arrays "**cal_curves**", "**a2d_bottom**", "**a2d_top**", "**a2d_amber**", "**a2d_red**", "**digital_bottom**", "**digital_top**", "**digital_amber**", and "**digital_red**" in the HEAP record "**remote_status_record.parm_data**" and in the array "**cn_record**" all defined below. The upper bound of this subrange is based on the value of constant "**max_linkends_per_remote**", defined above.

The subrange "**link_2state_ordinal**" is used as the index range for the arrays "**digitals**" in the global data record "**unpacked_response_record**", "**equip_digital**" in the Configuration data base record "**equipment_record**" and "**current_digitals**" in the HEAP record "**current_link_status_record**" all defined below. The upper bound of this subrange is based on the value of constant "**max_2states_per_link**", defined above.

The subrange "**a2d_ordinal**" is used as the index range for the arrays "**a2ds**" in the record "**unpacked_response_record**", "**equip_a2d**" in the Configuration data base record "**equipment_record**" and "**current_a2ds**" in the HEAP record "**current_link_status_record**", all defined below.

The subrange "**digital_ordinal**" is used as the index range for the arrays "**digitals**" in the record "**unpacked_response_record**", "**equip_digital**" in the Configuration data base record "**equipment_record**", and "**current_digitals**" in the HEAP record "**current_link_status_record**", all defined below.

The subrange "**master_crt_ordinal**" is used as the index range for the arrays "**cnt_cmds**" in the record "**statz_record**", "**crt_ptr**" in the Configuration data base record "**master_record**", and "**current_crt**" in the HEAP.

The subrange "**segment_remote_ordinal**" is used as the index range for the arrays "**transmission**" in the record "**statz_record**", "**remote_info**" in the Configuration data base record "**segment_record**", "**remote_status**" in the HEAP record "**segment_status_record**", and both HEAP arrays "**lats**" and "**lons**" which are fields in the record "**current_crt.latlons**" defined below.

The subrange "**expression_ordinal**" is used as the index range for the array "**expression**" in the record "**combo_record**", which is part of the record "**equipment_record**" defined below.

```
     data_control_block = ARRAY[1..144] OF INT;
```

The TYPE "**data_control_block**" defines the standard size File Manager (FMGR) data control block used to control FMGR disc file I/O. The FMGR data control block is discussed in the RTE-6/VM Programer's Reference Manual, p. 3-15. This TYPE definition is used by program DT, which must make extensive use of the FMGR I/O functions to transfer data from master to master. Use of standard Pascal I/O routines in program DT was not possible with the distributed system (DS) software. The DS software allows FMGR routines to be performed remotely (at the far node). The references in program DT are at lines 77, 106, 110, 114, 118, 121, 125, 129, and 132. This TYPE is also referenced by the routines in $TRLIB that are used to schedule type 6 programs. These references are at lines 127 (open_file), 131 (idrpl), 135 (close_file), 812 (run_prog) and 823 (clone_and_run).

```
     parm_array        = ARRAY[1..5] OF INT;
     time_array        = ARRAY[1..6] OF INT;
     twenty_int        = ARRAY[1..20] OF INT; {Ref by CMMD}
     hist_array        = ARRAY[1..nbr_bins{16}] OF INT;
     atoi_result       = ARRAY[1..33] OF INT; {Ref by AL, CC, CF, CN}
```

The TYPES above are grouped together because they are all commonly used arrays of single word integers. The most commonly used integer array above is the "**parm_array**", which is used primarily as the global storage for the five run string parameters passed to a program when it is scheduled. The INCLUDE module has a VAR called "**parms**" which is of TYPE "**parm_array**". Most TRAMCON programs call the routine "**get_parms**" ($TRLIB) with "**parms**" as the only parameter.

The integer array TYPE "**time_array**" is used by the time-synchronization programs SR, TIMPAS, and TIMSET to contain the six one-word integers that represent the current time/date.

The integer array TYPE "**hist_array**" is used later in this module to define the arrays "**hist_a2d**" and "**hist_digital**" in the HEAP record "**current_link_status_record**" and to define the array "**cal_curves**" in the HEAP record "**parm_record**". The TYPE "**parm_record**" is also used as the record definition for the disc file (CURVE. The analog and digital parameter data collected from the communications equipment is accumulated into discrete BINS. The TYPE "**hist_array**" has one cell for each of these discrete BINS with the upper limit of the array index controlled by the previously-defined constant "**nbr_bins**". The actual parameter value read is compared with the values in the array "**cal_curves**" and, when a match is made, the corresponding value in the array "**hist_a2d**" or "**hist_digital**" is incremented. These counts accumulate for one hour, then on the hour they are archived to the disc file (HIST.

The integer array TYPE "**atoi_result**" is used by the programs AL, CC, CF, and CN to hold the results of the ASCII string to the integer-parsing routine PARSE (system name $PARS), discussed in the RTE-6/VM Relocatable Library Reference Manual, p. 5-12.

NOTE

PARSE - Finding the information on this routine is difficult since there is NO mention of PARSE or its system library name **$PARS** in the index or any lists of routines that can be found in the Relocatable Library Manual. The only mention of it is in the table of contents, and this can be easily overlooked because of the large number of routines in the library. Also, this routine is completely omitted from the newer editions of the Relocatable Library Manual. The manual that corresponds to the A.85 version of software used to develop the TRAMCON software has part no. 92084-90013 and a December 1981 printing date.

```
response_data_types = (two_state, a2d, pulse_count); {3}
```

NOTE

There is one reference to this type definition in the routine "**transform_ordinal**" in $MPLIB.

```
msg_status = (polls, msg_ok, par_err, bad_res, no_ans); {5}
statz_record = {650 wds, record for file (STATZ}
  RECORD
  cnt_cmds: ARRAY[un..us{46}, master_crt_ordinal{5}] OF INT;{230 wds}
  transmission:
    ARRAY[master_segment_ordinal{4}] OF
      ARRAY[segment_remote_ordinal{21},msg_status{5}] OF INT; {420 wds}
  END;
```

The record TYPE "**statz_record**" describes some TRAMCON performance statistics that can be gathered to help the system developers fine-tune the software to make TRAMCON a more useful, more responsive product. The contents of the (STATZ disc file is placed in the HEAP record "**statz**" by the program INIT at system bootup. During any hour, the statistical data is accumulated into this HEAP record. The information in the HEAP "**statz**" record is permanently stored on disc in file (STATZ by the program HR every hour on the hour to ensure that, in case of trouble, the data will be no more than one hour old when the system is restored.

The first set of data collected, "**cnt_cmds**", is a count of the number of times that each TRAMCON command is entered. These counts are individually tallied for each command entered on each terminal keyboard. Currently, counts are tallied for the commands between and including the commands "**un**" and "**us**" only. Deleting or adding a command between these two commands will change the size of the record for file (STATZ and require that the new file size be written to the file before the TRAMCON software can run (see

Section 11.4). The intention of this data is not to spy on the operators, but to determine which commands are useful and which are not.

The second set of data collected, "**transmissions**", is a count of remote unit responses received from each remote unit on each monitored segment. The count is broken down into one of five categories that are enumerated in the CLASS TYPE "**msg_status**" above. Each response received is counted as one of the following:

OK - NO errors detected in response transmission
PE - the response contained at least one parity error
BR - the response was marked as a bad response
NA - the response was marked as No Answer, POLLER ONLY

Also available, only for a master in POLLER mode, is the total number of POLLs. This value is incremented each time a POLL message is sent to a particular remote unit. The intention of these data is to monitor the performance of the communications channel between the TRAMCON master computer and the remote units. To this day, this channel, the radio supervisory, is plagued with noise and transmission problems. These data could help solve those problems.

```
pcm_histogram_array = {200 wds}
  ARRAY[1..2,0..max_trunks_per_segment{100}-1] OF INT;
```

The array TYPE "**pcm_histogram_array**" is used in the definition "**pcm_histogram_record**" and as the field "**pcm_counts**" in the HEAP record "**segment_status_record**". This array holds the DIGROUP alarms for each end of each TRUNK (up to "**max_trunks_per_segment**") defined on a given segment.

```
pcm_histogram_record = ARRAY[0..23] OF pcm_histogram_array; {4800 wds}
```

The array TYPE "**pcm_histogram_record**" is the record description for the disc file (PHIST. One record contains the DIGROUP alarms for all the DIGROUP alarm counts for one TRAMCON segment for 24 hours. The programs HR and PH reference this TYPE for accessing the disc file (PHIST.

Program HR reads each record from file (PHIST and copies the values for the current hour from the HEAP value "**heap^.segment_status[segord].pcm_counts**" into the proper place in the 24-hour record, then rewrites the entire record to disc and sets the values in the HEAP to zero for the next hour.

Program PH reads the record for the selected segment and displays the values for the past 23 hours. The current hour's data is displayed from the HEAP "**pcm_counts**" array.

```
sc_indexs_record = ARRAY[0..29] OF {630 wds, record for disc file (SC }
                    RECORD passwd:two_chars; {1 wd}
                    f_description:forty_chars {20 wds}
                    END;
```

```
alfa_int_record = {2 wds}
  RECORD
  CASE data_type:BOOLEAN OF
   TRUE: (intgr: INT);
   FALSE: (alfa: two_chars)
  END;
```

The record TYPE "**alfa_int_record**" is used, much like the equivalence feature in FORTRAN, to refer to a value sometimes as a one-word integer (using identifier "**intgr**") and other times as two ASCII characters packed into one word (using identifier "**alfa**").

> NOTE
>
> Even though either definition for the body of the record, "**intgr**" or "**alfa**", is one word, the size of an "**alfa_int_record**" is two words. The overhead word is used to hold the value of the VARIANT TAG "**data_type**". The TAG field is optional and could be eliminated to avoid the one-word overhead.

```
eqt5_word = {1 wd}
  PACKED RECORD
  controller_availability: 0..3;
  eqt_type_code: six_bits;
  eqt_status1: 0..7;
  data_set_NOT_ready: BOOLEAN;
  eqt_status2: nibble;
  END;

eqt4_word = {1 wd}
  PACKED RECORD
  dma, auto_buffer, driver_do_pf, driver_do_to, timedout: BOOLEAN;
  subchannel: five_bits; select_code: six_bits;
  END;

status3_word = {1 wd}
  PACKED RECORD
  device_down: BOOLEAN; {1 bit}
  fill: nine_bits; subch: six_bits
  END;
```

The three record TYPEs "**eqt5_word**", "**eqt4_word**", and "**status3_word**" defined above are used to hold the one-word values returned by the system routine EXEC when an I/O status request (function code = 13) is made (refer to RTE-6/VM Programer's Reference Manual, p. 2-74). These status EXEC calls are only done to check the condition of the terminal devices since the devices are the only peripherals that can be removed and/or installed without disrupting the TRAMCON function. That is, terminal equipment may be defined now and not installed until a later date, or a terminal may break and be sent for repair while the TRAMCON master remains fully operational (refer to Section 9).

Each of the above values is one word long. The first parameter, "eqt5_word", is mandatory and contains the information in word 5 of the Equipment Table entry for the equipment number specified in the EXEC call. The other two values are optional. The content of the Equipment Table entries is detailed in the RTE-6/VM Programer's Reference Manual, pp. E-1 through E-6.

The routine "crt_status_check" in $TRLIB issues a status request on a specified terminal and checks for a "status3.device_down" indication or a "eqt4.timedout" indication. Either of these flags is interpreted as a malfunction of the terminal device and the program UP is informed of this fact. Program CMMD issues a status request for a terminal device, after getting input from an operator at the given terminal, to see if the keyboard input request has timed out. If "eqt4.timedout" is true, CMMD assumes that the operator no longer wishes to enter a command, the "Enter Command" prompt is erased and a new single character keyboard "wakeup" read request is issued. Program UP also issues a status request to a terminal device to see if the terminal is once again operable.

```
date_record = {37 words}
  RECORD
  yr, jdy, offset, heap_class_no, configuration_flag,unused: INT;{6 wds}
  version_date: INTEGER; {2 wds}
  version_nbr: REAL; {2 wds}
  segnames: ARRAY[master_segment_ordinal{4}] OF six_chars; {12 wds}
  nremotes: ARRAY[master_segment_ordinal{4}] OF INT; {4 wds}
  dmy1,dmy2,dmy3: INT; {3 wds}
  password: parm_array; {5 wds}
  time_serial_number: INT; {1 wd}
  message_serial_number: ARRAY[1..10] OF INT; {1 wd}
  logoff_class_no: INT; {1 wd, STOFF sets to 0 when ST command entered}
  END;                  {AUTOR checks if > 0 then TRAMCON is active}
```

The record TYPE "**date_record**" was so named because the time/date was the first information to be stored in this record. Since that time, a potpourri of information has made its way into this record. This record TYPE is a description of the record for the disc file (DATE. Basically, TYPE contains information about the TRAMCON master computer, which can be used by programs that are not scheduled by the program CMMD or by remotely scheduled programs in the rare instance when a TRAMCON master, including IPC, is operational but the TRAMCON On-Line software is not.

The year "**yr**" and the Julian day "**jdy**" are updated from the hardware clock by the program SETDT, which must be explicitly run by the operator whenever the hardware clock is adjusted. On most power failures, the hardware clock remains correct because it is backed up with a 6-volt lantern battery. If the lantern battery becomes too weak to back-up the clock and the system experiences a power failure, the hardware clock will fall behind for the remainder of the power outage.

The I/O CLASS number "**heap_class_no**" is a very important value, which allows TRAMCON programs that are NOT scheduled by the program CMMD to access the shared data area called the HEAP. This CLASS number is allocated by the program INIT at TRAMCON bootup. An output buffer containing the two-word first word address (FWA) of the HEAP is attached to this CLASS number. Any program NOT scheduled by CMMD but wanting to access the HEAP information can read the "**date_record**" from file (DATE, place the "**heap_class_no**" into global VAR "**parms[1]**" and call routine "**allocate_EMA**" which uses the value in "**parms[1]**" to perform a CLASS GET of the FWA for the HEAP.

The integer "**configuration_flag**" is another important item that indicates the availability of the fallback and/or new Configuration data bases. If this value is one, both a new and a valid fallback data base exist. Program SETDT sets this value to one. If the value is two, only a fallback data base exists; if the value is three, only a new data base exists. If the value is negative, the data bases are being updated and cannot be accessed temporarily.

The "**version_date**" and the "**version_nbr**" are set by the program SETVE and represent the On-Line version time/date stamp (seconds since midnight January 1, 1970) and version number (e.g., 1.82). These values are read and displayed in the lower lefthand corner of the TRAMCON logo at bootup by the program INIT. They are also displayed on the command line when the operator enters the VE command.

The arrays "**segnames**" and "**nremotes**" are set by the program INIT as the data base is read in at bootup. The arrays represent the short segment names and number of remote units for each segment defined in the data base. This information is currently used by the program DT running on another master. Program DT must determine what TRAMCON information the two communicating masters have in common and whether the TRAMCON On-Line software is running on the far master. To discover this, the DT program establishes contact with the selected master and reads the "**date_record**" from the (DATE file on the far master. The "**segnames**", "**nremotes**", and "**logoff_class_no**" are returned to the calling master. If "**logoff_class_no**" from the far master is non-zero, then the TRAMCON On-Line software is running on the far master. The DT program running on the calling master can then compare the segment names in the "**segnames**" data just received against its own segment names in the HEAP to determine if there is any TRAMCON data in common.

The values "**time_serial_number**", "**message_serial_number**", and "**offset**" are set and referenced by the programs involved in synchronizing the time/date clocks between masters.

The "**logoff_class_no**" is allocated and placed here by the program CMMD at system bootup. This integer is normally used as the CLASS number in the programs LOF and X to return a terminal device from session monitor to the TRAMCON On-Line software. When TRAMCON is terminated with the ST command at the system console, "**logoff_class_no**" is set to zero by the program TROFF.

Two programs are interested in the situation when the TRAMCON On-Line software is inactive but the master computer is functioning (this is a very unusual situation). Program AUTOR, which recovers from power failures, must know if the on-line software was NOT running so that it does not attempt to reschedule some of the periodic programs, such as HR. Also, the DT program at a distant master must know that the On-Line software at the local master is NOT running so that it does not attempt to transfer TRAMCON related data.

```
msg_record = {16 wds}
  RECORD
  caller_class, seg_ord, remote_ord, msg_len: INT; {4 wds}
  cmd_byte, cat_byte: CHAR; {2 wds}
  msg_body: twenty_chars {10 wds}
  END;
```

The record TYPE "**msg_record**" is used by any program that wants to send a request to a particular remote unit on a particular segment. Currently these programs include PLRP, MTRP, and SW. These messages are composed by the program sending the request and passed, via a CLASS WRITE/READ request issued on the CLASS number "**poll_class**" (found in the HEAP) with LU set to zero (see RTE-6/VM Programer's Reference Manual, p. 2-38), to the program POLL, which acquires the messages with a CLASS GET request issued on the same "**poll_class**". Program POLL acts as a central clearing agent for all outgoing messages to the segments and their remote units. This ensures that all messages issued for each segment channel are placed in sequential order and that no two messages are issued for the same channel at the same time.

Each program (currently only PLRP, MTRP, and SW) has a unique CLASS number (allocated by program CMMD and stored in the HEAP at bootup). Each program looks for input (remote unit responses) via a CLASS GET on that CLASS number only. That CLASS number is passed to program POLL as the value of "**caller_class**" and POLL issues a CLASS READ (EXEC 17) on that "**caller_class**" so that the proper program gets the remote unit response. The segment and the remote unit on that segment are specified to POLL in the values "**seg_ord**" and "**remote_ord**", where "**seg_ord**" is in the range "**master_segment_ordinal**" and "**remote_ord**" is in the range "**segment_remote_ordinal**", both of which are described above in this section. The length of the message going to the remote unit, in bytes, is specified in "**msg_len**". The request type is specified in "**cmd_byte**". Current request types supported for the DATALOK10 are

cmd_byte	Request Description
N	**NORMAL Poll with FULL response**
C	**Relay CONTROL request with FULL response**

Program SW sends a "C" request and programs MTRP and PLRP send "N" requests.

```
entry_point_record = PACKED RECORD {4 wds}
                     ep_name: five_chars;  ep_type: seven_bits;
                     ep_address: INT
                     END;

disc_block_buff = ARRAY[1..32] OF entry_point_record; {128 wds}
disc_addr_rec = {1 wd}
  PACKED RECORD track: nine_bits; sector: six_bits; odd: BOOLEAN END;
```

The three record descriptions above can be used by any program to locate any system ENTRY POINT that is not stored in the memory-resident portion of the operating system. Due to space restrictions, several seldom-used ENTRY POINTS are stored in a table in the disc-resident portion of the operating system. This ENTRY POINT table is described in the RTE-6/VM Technical Specifications Manual, Appendix H. Given an entry point name, a program searches through this table of 4-word entries looking for a match with "**ep_name**". If a match is found, the corresponding memory address in "**ep_address**" can be used to acquire the value of the entry point.

```
pf_record = {12 wds, record for disc file (PF }
  RECORD
  on_year ,on_jday ,on_hour ,on_minute ,on_sec ,on_msec,
  off_year ,off_jday ,off_hour ,off_minute ,off_sec ,off_msec: INT
  END;
```

The record TYPE "**pf_record**" is the record definition for the disc file (PF that contains all the power failure event messages. These messages are composed and placed in file (PF by the program AUTOR each time power is restored. The time of failure is acquired by issuing a read request on the power fail LU and the time of recovery is read from the software clock after it has been updated from the hardware clock. The time/dates are stored in the six one-word integer form that is returned from the EXEC 11 request. The duration of the power failure is calculated by subtracting the OFF time/date from the ON time/date.

```
crt_msg_record = {37 wds}
  RECORD
  msg_ord: INT;     {1 wd, -1 if message being passed}
  msg_crtord: INT; {1 wd, -1 if broadcast}
  msg_segord, msg_remoteord, msg_audible,
    msg_priority, msg_length: INT; {5 wds}
  msg_alf: sixty_chars {30 wds}
  END;
```

Each terminal (CRT) defined on this master has a set of messages that are to be displayed on the command line of the CRT. Any program that wants to add or delete a message from the list for a particular CRT or for all CRTs must compose a "**crt_msg_record**" and pass it to the CRT message-processing program MSG via a CLASS WRITE/READ request with LU set to zero (see RTE-6/VM Programer's Reference Manual, p. 2-38) and issued on the CLASS number "**msg_class**", found in the HEAP.

If the request references an existing message (such as a delete request), the value of "**msg_ord**" will be a valid index into the arrays pertaining to the CRT messages in the HEAP record "**current_crt**"; otherwise, "**msg_ord**" is set to -1. The CRT to which this message belongs is identified by the value of "**msg_crtord**", which has the range "**master_crt_ordinal**" and is used as an index into the HEAP array "**current_crt**". This item is set to -1 if the message is intended for ALL (broadcast) CRTs. An example of a broadcasted message is the "**XXXXXX Segment is NOT Responding**" message issued by the program MTRP if NO response from any remote unit on the entire XXXXXX segment has been received for one minute. Many messages, such as the previous example, make reference to particular segments and/or remote units. These segments and remote units are identified by the values "**msg_segord**" and "**msg_remoteord**". If the audible alarm should accompany a message, it is defined by the value of "**msg_audible**". The messages for each CRT are prioritized by the value of "**msg_priority**" so that if more than one message is queued for a given CRT, the highest priority message will be displayed. The actual message length, in bytes, is passed in "**msg_length**". Finally, if it is a new message, the message body is passed in the string "**msg_alf**".

```
data_char_type = (ASCII_char, eight_b, int_ger, six_b, BCD);
data_char = {1 wd}
  PACKED RECORD
  CASE char_tag: data_char_type OF
   ASCII_char:(ch: CHAR);
   eight_b: (bits: PACKED ARRAY[0..7] OF BOOLEAN);
   six_b: (sixbits_fill: 0..3; sixbits: six_bits);
   int_ger: (intgr: INT);
   BCD: (BCD_filler: four_bits; BCD_value: four_bits)
  END;
```

The record type "**data_char**" can be used by any program that needs to deal with a one-word data item in different formats at different times. For example, the remote unit response is defined as a PACKED string of CHARS and is read in that format. When unpacking the two-state information, the "**sixbits**" variation of the "**data_char**" definition is used because the two-state data are packed as six one-bit pieces of data in the lower six bits of the byte. To unpack the analog-to-digital information, which is encoded as binary coded decimals, the "**BCD_value**" variation is used.

```
alarms_array = PACKED ARRAY[category_ordinal{4},
                         link_2state_ordinal{144}] OF BOOLEAN; {48 wds}
a2ds_array =  ARRAY[category_ordinal{4},a2d_ordinal{6}] OF INT;{24 wds}
digitals_array = ARRAY[category_ordinal{4},digital_ordinal{22}] OF INT;
unpacked_response_record = {160 wds}
  RECORD
  alarms: alarms_array; {48 wds}
  a2ds: a2ds_array; {24 wds}
  digitals: digitals_array {88 wds}
  END;
```

The "**unpacked_response_record**" is the generic format for a remote unit response. The remote unit response preprocessing routines "**transform_ordinal**" and "**unpack_response**" in $MPLIB convert responses received from each type of remote unit supported (currently there are two models of the DATALOK10, 1D and 1E) and converts those uniquely formatted responses into the generic format of an "**unpacked_response_record**". The response processing routine "**process_response**" in $MPLIB then analyzes all responses in this generic format regardless of what type of remote unit it came from. The data in any response are divided into three types: (1) "**alarms**", the two-state values; (2) "**a2ds**", the analog voltages; and (3) "**digitals**", the pulse counts. The generic response is equivalent to a Configuration data base equipment record multiplied by the number of categories per PHYSICAL remote unit. That is, this record describes an entire PHYSICAL remote unit response while an EQUIPMENT record describes only one category (one link end) of a response. With this data structuring scheme each category of a PHYSICAL remote unit could monitor a different kind of communications equipment.

The array TYPES "**alarms_array**", "**a2ds_array**", and "**digitals_array**" are referenced in the definition of "**unpacked_response_record**" directly below them and as the TYPES of the global VARs "**init_2states**", "**init_a2ds**", and "**init_digitals**" in the global VAR INCLUDE module [MPVAR. These three VARs are used by the routine "**unpack_response**" to clear out global VAR "**unpacked_response**" before unpacking each response.

The sizing of the "**unpacked_response_record**" is controlled by the constants "**max_2states_per_link**", "**max_linkends_per_remote**", "**max_a2ds_per_link**", and "**max_digitals_per_link**", defined above.

```
     si_response_record =    {172 wds, Simulator response record, file (RR }
       RECORD
       request_error: INT; {1 wd, nonzero if remote unit detected error in
                                  request received from TMT.
                                   1. msg length limit exceeded.
                                   2. command error.
                                   3. category error.
                                   4. number(s) out of range.
                                   5. date-time error.
                                   6. numbers NOT in ASCENDING order.
                                   7. numbers duplicated.
                                   8. count error.
                                   9. action error.
                                  10. unwired/unused error.
                                  11. momentary control deactivation error.
                                  12. configuration table error. }
       diag_error: INT;    {1 wd, nonzero if remote unit background
                                 diagnostics discover an error.
                                   1. main processor failure.
                                   2. data acquisition failure.
                                   3. memory board failure.
                                4-17. I/O card failure.
                              18-255. software fault. }
```

```
        diags: twenty_chars; {10 wds, one char per module in remote unit.
                                      "o" - no CPU fault identified.
                                      "m" - main CPU fault.
                                      "a" - auxiliary CPU fault. }
        response_body: unpacked_response_record; {160 wds}
        END; {si_response_record}
```

The record "**si_response_record**" is used by the response simulation program SI to define the packaged remote unit responses, which it stores on file (RR.

```
      archive_alarm_status_record = {119 wds}
        RECORD
        arch_year: INT;
        arch_jday: nine_bits;
        arch_hour: seven_bits;
        arch_minute, arch_second: byte;
        archive_alarms: {114 wds}
           PACKED ARRAY[category_ordinal{4},link_2state_ordinal{144}] OF
             PACKED RECORD {3 bits}
             arch_just_cleared, arch_new_alarm,arch_alarm_on: BOOLEAN
             END;
        END;
      archive_idx_record = ARRAY[1..124] OF INT; {124 wds}
      archive_record = {125 wds, record for disc file (ARCH}
        RECORD
         CASE ar_rec_type: BOOLEAN OF
          FALSE: (arch_idx: archive_idx_record); {124 wds}
           TRUE: (arch_rcd: archive_alarm_status_record) {119 wds}
        END;
```

The record TYPE "**archive_record**" is used to describe records on disc files (ARCH and (ARCHX, which store the change-of-state events reported by each remote unit being monitored by the given master. The very first record in file (ARCH is an index into the rest of both files and, therefore, has a different record definition than the rest of the records. The two definitions for "**archive_record**" (one for the index and the other for an actual archive data record) are equated using a Pascal RECORD VARIANT. The "**arch_idx**" identifier refers to the index record definition "**archive_idx_record**" and is the definition of the first record in the file only. The identifier "**arch_rcd**" refers to the "**archive_alarm_status_record**", which is the definition of all the records on the file except record number one.

The index record, number one in file (ARCH, contains two one-word integer pointers for each remote unit monitored by the master. These integers indicate the next available archive record number in each file, (ARCH and (ARCHX, for the given remote unit.

NOTE

The index record is currently sized at 124 words. A more convenient size would be 128 words, since that is the disc block size. This would place all records in file (ARCH on disc block boundaries and make the I/O to file (ARCH as efficient as possible. The number of remote units supported by the archive file index would increase from 124 to 128, but both values are much larger than the current maximum number of remote units that can be monitored by one master which is 42 (2 segments of 21 remote units each). To use the full 128 words for the index record, the VARIANT TAG "**ar_rec_type**" should not be specified, to avoid incurring the one word of overhead needed to store the value of that tag.

NOTE

The fields "**arch_jday**", "**arch_hour**", "**arch_minute**", and "**arch_second**" are defined as TYPES that are smaller than one word (16-bits) in case the need arises to pack the archive record. This is currently NOT the case since the index record variant is the larger of the two definitions. Of course, access to these items is quicker in this unpacked state.

DS_node = nill{-1}..max_masters_per_net{4};

The network software purchased from HP to support the master-to-master communications over the InterProcessor Communications Channel (IPC) is referred to in the literature as distributed systems (DS). Each master on the DS network has a unique node number associated with it. The range TYPE "**DS_node**" specfies the range of these node numbers. This range TYPE is referenced in the definition "**alt_mast_array**" defined below. These node numbers are assigned to each master by the Configurator program when it composes the DS initialization program answer file (DINIT as part of the master specific data base. The node number for each master is derived from the record number of the corresponding master record in the universal Configuration data base file <MAST. That is, if the Donnersberg master record is the first record in file <MAST, then the DS node number for the Donnersberg master is 1. The disc file (DINIT is tailored for each particular master and is used by the program DINIT to initialize the DS software on the particular master at bootup time and to establish that master as a node on the IPC network.

dictionary_ptr = nill{-1}..dictionary_size{14000} - 1;

A "**dictionary_ptr**" is an index into the 14000-character string called the "**dictionary**", which is created by the Configurator, stored on disc file

(DICT, and read into the HEAP VAR "**heap^.dictionary**" at system bootup. Variables of TYPE "**dictionary_ptr**" are the index, in the character array "**heap^.dictionary**", of the first character of a dictionary word defined below.

dictionary = PACKED ARRAY [0..dictionary_size-1] OF CHAR;{14000 chars}

The dictionary is a continuous string of ASCII characters created automatically by the Configurator program as the operator creates other elements of the Configuration data base that have components that are themselves ASCII strings. The dictionary is currently 14000 characters long and that size is anticipated to be as long as it needs to be for any TRAMCON master. Figure 52 lists the items that are currently stored in the dictionary. The dictionary is stored in the data base file (DICT as one 14000 byte record and, like the rest of the Configuration data, is read and stored in the shared EMA area called the HEAP by the program INIT when the TRAMCON system is initiated. Because of system addressing limitations, records larger than 1024 words cannot be read directly into EMA. A local buffer, the size of the dictionary, is declared in INIT and the dictionary record is read into this buffer. From the local buffer, the data are transferred to the HEAP record "**heap^.dictionary**". Refer to Section 4.1 for the details of TRAMCON initialization.

dictionary_word = PACKED ARRAY[1..dictionary_word_size{30 chrs}]OF CHAR;

The dictionary string defined above is separated into substrings, called WORDS, by the ASCII delete character. The "**dictionary_word**" defined above is limited to a maximum length of 30 characters ("**dictionary_word_size**" defined above). These words are accessed by the routine "**read_dict**" (refer to Section 8.2.4.1) by using a variable of TYPE "**dictionary_ptr**" as the location in the "**dictionary**" of the first character of the word and using the next ASCII delete character found as the end of the word.

dictionary_record_ptr = ^dictionary;

There is only one VAR in the TRAMCON software that is declared TYPE "**dictionary_record_ptr**" and that is the field in the HEAP record "**heap_ptrs**" called "**dictionary**" defined below. A "**dictionary_record_ptr**" is a two-word EMA (HEAP) address that points to the one and only Configuration data base dictionary stored in EMA. Most programs use the routine "**read_dict**" in $TRLIB to retrieve data from the dictionary. Routine "**read_dict**" uses the global VAR "**heap^.dictionary**" as the first word address (FWA) of the dictionary.

Site Code	Alarm Names	Comm Equipment Names
Site Names	Status Names	remote unit Equipment Names
Country Names	Parameter Units	Trunk IDs
Service Branch Names	Relay Status Names	Short Segment Names
Parameter Names	Relay Names	Long Segment Names

Figure 52. Items in data base that are DICTIONARY WORDs.

```
site_record = {9 wds}
  RECORD
  site_code: dictionary_ptr; {1 wd, 3 letter code}
  site_name: dictionary_ptr; {1 wd, up to 18 letters}
  master_flag: BOOLEAN; {1 wd}
  country: dictionary_ptr; {1 wd}
  latitude, longitude: REAL; {4 wds}
  service_branch: dictionary_ptr {1 wd}
  END;
site_record_ptr = ^site_record;
```

The record TYPE "**site_record**" describes the Configuration data base site records that are read from disc file (SITE into the HEAP by program INIT at bootup. References in other data base records to these site records are converted from their data base integer value (representing a site record number in file (SITE) to a two-word HEAP address of TYPE "**site_record_ptr**". The fields in the "**site_record**", that are of TYPE "**dictionary_ptr**", are integer values interpreted as indices in the character array "**dictionary^**".

```
name_list = ARRAY [0..max_words{6} - 1] OF dictionary_ptr; {6 wds}
```

Several items in the Configuration data base records defined below are phrases composed of dictionary words. The TYPE "**name_list**" defines a phrase composed of up to "**max_words**" (currently set to 6) many "**dictionary_word**"s. Examples of items in the data base that are dictionary phrases are

```
equipment_record
  alarm_name   Phrase describing an alarm or status
  param_name   Phrase describing an analog or digital parameter
  relay_name   Phrase describing a relay switch
  combo_name   Phrase describing a combination two-state alarm
linkend_record
  name         Phrase describing a specific alarm
```

```
two_state_record = {7 wrds}
  PACKED RECORD
  alarm_name: name_list; {6 wds}
 {Last word of names is indicated by nill in the next dictionary_ptr
  entry or end of array.}
  alarm_type: nibble; {4 bits, 0 = status, 1 = minor, 2 = MAJOR alarm}
  specific_name_flag: BOOLEAN;{if this alarm has a link specific name}
  pcm_port: byte {8 bits}
  END;
```

The record TYPE "**two_state_record**" is used as the TYPE of the field "**two_states**" in the "**equipment_record**" defined below. That is, this record defines the two-state portion of the Configuration data base equipment record. One of the more unusual features of the two-state data has to do with specific names.

```
parameter_record = {8 wds}
  RECORD
  param_name: name_list; {6 wds}
  param_type: byte;  {which function handles the type of parameter
                      involved,as non-calibrated,calibrated,count data}
  param_units: dictionary_ptr
  END;
```

The record TYPE "**parameter_record**" is used as the TYPE of the fields "**equip_a2d**" and "**equip_digital**" in the "**equipment_record**" defined below and is referenced in the program PA. That is, this record defines the analog and digital parameter portions of the Configuration data base equipment record. The field "**param_type**" is used by the response-processing routine "**process_response**" ($MPLIB), which calls routine "**parm_def**" ($MPLIB) to establish a set of characteristics for any given parameter. The parameter types are established by the Configurator and each refers to a unique combination of the characteristics. The currently supported parameter types are listed in Figure 53.

```
relay_record = {10 wds}
  RECORD
  relay_name: name_list; {6 wds}
  relay_type: byte;  {1 wd, latching or nonlatching}
  relay_status: INT; {1 wd, ordinal into "two_states" locating
                       corresponding status bit}
  open_name,         {ASCII word for open state e.g., on-line}
  closed_name: dictionary_ptr {1 wd}
  END;
```

The record type "**relay_record**" describes the remote relay section of the Configuration Data Base EQUIPMENT record. Each possible relay for an EQUIPMENT CATEGORY has an English name, "**relay_name**", and a type (described in Figure 53) assigned to it by the Configurator. An attempt is made to allocate a nonlatching status indicator for each relay defined ("**relay_status**"). This status bit should indicate the results of the particular relay switch function. For example, the relay to switch receiver A ON or OFF Line should be associated with the status indicator "**Receiver A Operating**". The last two fields, "**open_name**" and "**closed_name**", are English names that identify the OPEN and CLOSED states of the relay.

TYPE	decimal_places	decreasing	calibrate	two_sided_th	Examples
1	0	TRUE	TRUE	nill	DRAMA RSL
2	2	TRUE	FALSE	nill	DRAMA Sig Qual
3	2	TRUE	TRUE	5000	Site Battery
4	0	FALSE	TRUE	nill	FRC162 RSL, BDM
60	0	FALSE	FALSE	nill	Digital
61	0	FALSE	FALSE	nill	MD-918 #1 Err Rate
62	0	FALSE	FALSE	nill	MD-918 #2 Err Rate

Figure 53. Transmission parameters currently supported.

```
exp_tree_node = {3 wds}
  RECORD
  op: INT; {op is an operator with values 1=.AND., 2=.OR., and 3=.NOT.}
  left_link, right_link: INT {2 wds}
  END;
```

Each node in a LOGICAL expression tree defined below consists of a LOGICAL operator "**op**" (with possible values of AND, OR, or NOT) and one or two operands, "**left_link**" and/or "**right_link**". If the operand is positive, its value is an index into the array "**two_states**" in the Configuration data base equipment record "**equipment_record**" (see below). If the operand is negative, the absolute value of the operand is an index into the same "**expression_tree**". That is, it points to a subexpression. If the operator "**op**" is the unary operator .NOT., then there is only one operand and it is in the right link.

```
expression_tree =ARRAY [expression_ordinal{16}]OF exp_tree_node;{48 wds}
```

An "**expression_tree**" is an array of binary nodes each of TYPE "**exp_tree_node**" defined above. These "**expression_trees**" are used to define combination alarms explained below. These trees are evaluated by the recursive routine "**evaluate_node**" in library $MPLIB. The only field in the TRAMCON data defined of TYPE "**expression_tree**" is the field "**expression**" in the record "**combo_record**" below. The record "**combo_record**", in turn, is part of the Configuration data base record "**equipment_record**".

```
combo_record = {55 wds}
  RECORD
  combo_name: name_list; {6 wds}
  combo_type: BOOLEAN;  {MAJOR = TRUE or minor = FALSE}
  expression: expression_tree {48 wds}
  END;
```

The record type "**combo_record**" describes the combinatorial two-state section of the Configuration Data Base EQUIPMENT record. Each possible combinatorial for an EQUIPMENT CATEGORY has an English name, "**combo_name**", and a type, "**combo_type**", assigned to it by the Configurator. A combinatorial is a fabricated, rather than real two-state alarm composed of two or more individual two-states related logically. The combinatorial is defined by the field "**expression**".

```
equipment_record = {2313 wds}
  RECORD
  equipment_name: dictionary_ptr;
  two_states:
    ARRAY [link_2state_ordinal{144}] OF two_state_record;{1008 wds}
  equip_a2d: ARRAY [a2d_ordinal{6}] OF parameter_record; {48 wds}
  equip_digital: ARRAY [digital_ordinal] OF parameter_record; {176 wds}
  relays: ARRAY [0..max_relays_per_link{20}-1] OF relay_record;{200 wds}
  combos: ARRAY [0..max_combos_per_link{16}-1] OF combo_record {880 wds}
  END;
equipment_record_ptr = ^equipment_record;
```

The record TYPE "**equipment_record**" describes the Configuration data base equipment records that are read from disc file (EQT into the HEAP by program INIT at bootup. References in other data base records to these equipment records are converted from their data base integer value (representing an equipment record number in file (EQT) to a two-word HEAP address of TYPE "**equipment_record_ptr**". The fields in the "**equipment_record**", that are of TYPE "**dictionary_ptr**", are integer values interpreted as indices in the character array "**dictionary^**" elsewhere in the HEAP.

The term "**equipment**", here refers to the communications and/or site hardware that is being monitored. Figure 54 lists all the equipment currently monitored by the TRAMCON on-line software. For each of the unique kinds of equipment listed in Figure 54 there is a corresponding equipment record in the Configuration data base file (EQT.

1. **Site e.g., Battery voltage, Tower Light, Door**
2. **FRC-171 DRAMA**
3. **FRC-80 and Siemens 120-6000**
4. **FRC-177 Troposcatter**
5. **FRC-162/165**
6. **FRC-113**
7. **Codenoll Fiber**
8. **Northern Telecom/Collins**
9. **Fiber/DNI**
10. **FAC-3 Fiber**

Figure 54. Transmission equipments currently monitored.

The types of data that can be monitored are separated into "**two_states**", "**equip_a2d**", "**equip_digital**", and "**combos**", where "**combos**" refers to alarms derived from combinations of two-state alarms reported by the monitored equipment. That is, the combination alarms are NOT reported by the equipment directly. Instead, the TRAMCON On-Line software creates these alarm occurrences based on logical combinations of real alarms and status values. The equipment functions that can be remotely activated are defined in the array "**relays**".

```
specific_name_record = {7 wds}
  RECORD
  name: name_list; {6 wds}
  alarm_number: nill..max_2states_per_link{144} - 1 {1 wd}
  END;
```

The "**specific names**" feature allows general equipment definitions to be tailored for each installation. For example, there is one equipment record that describes the FRC-171 Drama radio. This record lists 16 digroup alarms. These alarms are identified in the "**two_state_record**" with a name like "**PCM1**", but further identification is needed that is specific to one site as

"AVO to CLO trunk". Any given installation may actually use from none, to all 16 of these alarms. The software determines whether a particular digroup alarm applies at any given location by examining the **"specific_name"** field in the linkend record. If the first word of the **"name"** field is not **"nill"**, the alarm is defined at this location and has a unique (or specific) name.

```
     link_def_ptr = -1..max_links_per_net-1;

     linkend_record = {565 wds}
       RECORD
       links_ptr: link_def_ptr; {1}
       specific_name: ARRAY[0..1,0..max_specific_names{40}-1] OF
         specific_name_record {560 wds}
       relay_exists: PACKED ARRAY[0..1,0..max_relays_per_link-1] OF BOOLEAN
       END;
     linkend_record_ptr = ^linkend_record;
```

The record type **"linkend_record"** describes the Configuration Data Base link end record found in file (LINK. Each set of communications equipment monitored by the given master has a corresponding **"linkend_record"** defined in the data base. This record contains all the information unique to this end of a given link. If any of the alarm or status indicators have names that are specific to this linkend, they are specified in **"specific_name"**. The number of alarm/status indicators that can be specific at any given linkend is limited by the subrange TYPE **"max_specific_names"** which is currently set at 40. The linkend specific feature was extended to apply to the remote-controll relays as well, with the addition of the **"relay_exists"** array. Here, however, the feature is limited to a one-bit BOOLEAN flag per relay that indicates whether a given relay is defined at this linkend or not. Any relay must use the same generic name wherever it is defined. On the other hand, all the relays for any linkend can be specifically defined at any linkend since this array is limited by the subrange TYPE **"max_relays_per_link"**. By adding a second dimension (0..1) to the specific data arrays, the specific information for both ends of a given link can be stored in the same linkend record. The information common to both ends of the link, such as the link ID and the communications equipment type, are found by tracing the **"links_ptr"** into the LINKS record.

```
     remote_types = (drama_pulsecom, frc165_pulsecom, IRU);
     remote_record = {15 wds}
       RECORD
       remote_polling_id: byte;
       site: site_record_ptr; {2 wds}
       site_equipment: equipment_record_ptr; {2 wds}
       linkend_info: ARRAY [0..max_linkends_per_remote{3} - 1] OF
                     linkend_record_ptr; {8 wds, end of data = NIL}
       remote_equip_type: remote_types; {1 wd}
       remote_equip_name: dictionary_ptr {1 wd}
       END;
     remote_record_ptr = ^remote_record;
```

The record type "**remote_record**" describes the Configuration Data Base remote unit record found in file (REMOT. Each remote unit monitored by the given master has a corresponding "**remote_record**" defined in the data base.

Each remote unit has a unique identification code, "**remote_polling_id**", which ensures that one and only one remote unit responds to each request for data from the master. This "**remote_polling_id**" is set by the Configurator and must correspond exactly with the hardware straps set by the installation team. The physical location of the remote unit is indicated by the field "**site**". The facility equipment (e.g., door, generator, tower light) monitored by the remote unit is defined by the field "**site_equipment**". The sets communications equipment monitored by the remote unit are specified in "**linkend_info**". The remote unit hardware is defined by the two fields "**remote_equip_type**" and "**remote_equip_name**". Currently, there are two types of remote units in use, the DATALOK10 Model 1D and the DATALOK10 Model 1E.

```
trunk_link_array = ARRAY[0..max_sites_per_trunk{18}-1] OF {36 wds}
                     RECORD {2}
                     trunk_links: link_def_ptr;
                     trunk_port: byte;
                     END;
trunk_record = {44 wds}
  RECORD
  trunk_id: dictionary_ptr; {1 wd}
  links_in_trunk: trunk_link_array; {36}
  last_node: INT; {1}
  trunk_ends: ARRAY[0..1] OF {4}
              RECORD
              end_site: site_record_ptr; {2}
              port: byte
              END;
  END;
trunk_record_ptr = ^trunk_record;
```

The record type "**trunk_record**" describes the Configuration Data Base trunk record found in file (TRUNK. Each communications trunk (i.e. DIGROUP or 24-channel group) monitored by the given master has a corresponding "**trunk_record**" defined in the Data Base. Each trunk has a unique identifier, "**trunk_id**", assigned to it by the military. Each trunk passes through a series of links which are specified in "**links_in_trunk**". The names of the locations of each end of a given trunk link can be found by tracing the pointer "trunk_links" into the LINKS record. The field "**trunk_port**" indicates the multiplexor port on which the given trunk exits the "from" site and enters the "to" site. The links information for the first and the last entry are repeated in the array "**trunk_ends**" by the program INIT at runtime for convenience. The value "**last_node**" is also set at runtime to indicate how many of the possible "**links_in_trunk**" entries are actually defined for each trunk.

```
set_of_remotes = SET OF segment_remote_ordinal; {2 wds}

remotes_array =
  ARRAY[master_segment_ordinal{4}] OF set_of_remotes; {8 wds}

segment_record = {244 wds}
  RECORD
  short_segment_name, long_segment_name: dictionary_ptr; {2 wds}
  remote_info:
    ARRAY [segment_remote_ordinal{21}] OF remote_record_ptr; {42 wds}
  trunk_info:
    ARRAY[0..max_trunks_per_segment(100)-1]OF trunk_record_ptr {200 wds}
  END;
segment_record_ptr = ^segment_record;
```

The record type "**segment_record**" describes the Configuration Data Base segment record found in file (SEG. Each TRAMCON segment monitored by the given master has a corresponding "**segment_record**" defined in the data base. Each segment has a short name, "**short_segment_name**", which is used extensively in the software to uniquely identify a given segment. Most TRAMCON commands allow the operator to specify the segment by entering the short segment name. For the software to make a match, the operator must spell the name exactly as it appears in the data base. This exact spelling of the short segment name and the rest of the information in these segment records can be displayed by entering the command "**SE**". The long name, "**long_segment_name**", is used for display only. The set "**currently_polled**" is initialized to all remote units on the given segment by the program INIT at bootup and is updated at run-time by program CMMD in response to either command "**PO**" or "**PM**" (see Section 5.4).

The array "**remote_info**" contains pointers to remote records for each remote unit defined on the given segment. Each remote record contains the configuration information for one remote unit. Each remote unit has a unique address, "**remote_polling_id**", used by the software to request data from that remote. The physical location information for the remote unit is pointed to by the site record pointer "**site**". The transmission equipment monitored by each remote is divided into "**categories**" with the first category being the site equipment and all other categories referred to as "**link-end**" categories. The equipment record describing the equipment monitored at the site is pointed to by the pointer "**site_equipment**".

```
comm_info_record = {1 wd}
  PACKED RECORD baud:byte; auto_answer, modem:BOOLEAN END;
crt_record = {6 wds}
  RECORD
  comm_info: comm_info_record; {1 wd}
  terminal_type: byte; {1 wd}
  printer_type: 0..7;  {1 wd, 0 = no printer}
  location: site_record_ptr; {2 wds}
  location_qualifier: CHAR {1 wd}
  END;
crt_record_ptr = ^crt_record;
```

The record type "**crt_record**" describes the Configuration Data Base terminal (CRT) record found in file (CRT. Each terminal device installed on a given master has a corresponding "**crt_record**" defined in the data base. The physical communications parameters, such as baud rate, modem, or hardwire connection, for each CRT are set by the Configurator in "**comm_info**".

The terminal type (2647F, 2627A or 2397A) is set in "**terminal_type**". This value is now overridden by the logon program LO when an operator signs on at any given terminal device. The terminal type is determined by reading the terminal ID from the actual device. The value for "**printer_type**" is also determined On-Line by reading the external device status from the terminal at sign on time. The terminal is physically located at the site indicated by "**location**". To distinguish between several terminals that might be at the same location, the "**location_qualifier**" was included.

```
    alt_mast_array =ARRAY[0..max_masters_per_segment{4}-1]OF DS_node;{4 wds}
    master_record = {47 wds}
      RECORD
      site_ptr: site_record_ptr; {2 wds}
      master_name_qualifier: CHAR; {1 wd}
      segment: ARRAY [0..max_segments_per_master(4) - 1] OF {32 wds}
                 RECORD {8 wds}
                 seg_ptr: segment_record_ptr; {2 wds}
                 segment_lu: INT; {1 wd}
                 poll_monitor: 0..2;{1 wd, 0=inactive, 1=monitor, 2=poller}
                 alternate_masters: alt_mast_array {4 wds}
                 END;
      crt_ptr: ARRAY [master_crt_ordinal{5}] OF crt_record_ptr; {10 wds}
      confi_version: INTEGER {2 wds}
      END;
    master_record_ptr = ^master_record;
```

The record type "**master_record**" describes the Configuration Data Base master record found in file (MAST. Each TRAMCON master computer has one "**master_record**" at the top of the Configuration data base hierarchy that is pointed to by the two-word EMA pointer "**master**", which is a field in the basic HEAP record "**heap**" described below. This means that "**master**" is the only item in the TRAMCON software that is of TYPE "**master_record_ptr**".

The fields in the "**master_record**" include a two-word EMA address, "**site_ptr**", which points to the site record that corresponds to the site at which the computer is physically located, along with a name qualifier, "**master_name_qualifier**", to distinguish between multiple masters located at the same site. The array "**segment**" contains the descriptions of all the TRAMCON segments defined for this master. This array is indexed by the Global VAR "**segord**", which is declared in the INCLUDE module [TRVAR (see Section 11.3). The pointer "**seg_ptr**" is a two-word EMA address that points to a "**segment_record**" which, in turn, describes a particular segment defined on this master. These "**segment_record**"s are described above.

NOTE

Certain LU numbers have been defined to point to the segment or polling channels on the TRAMCON master computer. The LU numbers are the same numbers in decimal as their corresponding select code (I/O slot) numbers are in octal. For example, LU 15 (decimal) is associated with I/O slot 15 (octal). There must be careful coordination between the data base Configurator and the TRAMCON installation driver to ensure that the responses for a given segment are reported on the correct I/O channel. If the drawing doesn't match the Configuration data base, the result would be NO answers at the polling master since the remote unit addresses are unique in theater. On a master that is in monitor mode for the mismatched segment, the result would be responses generated by polling messages from the other master, but responses from the wrong set of remote units.

The "**segment_lu**" is a one-word logical unit number assigned to a particular segment. This LU value can be viewed by entering the SE command.

The "**poll_monitor**" flag indicates the current status (poller or monitor) for each segment on this master. This status value is changed by the program PM in response to the PM command. The "**alternate_masters**" array indicates which other TRAMCON master computers are assigned to monitor the given segment and thus have the same segment defined in their Configuration data base. By design, this array must have at least one non-NIL entry for each segment. TRAMCON master computers are uniquely specified by using their IPC network node number. These node numbers are explained under "**network_record**" above.

The ARRAY "**crt_ptr**" contains one two-word EMA address for each terminal display device defined on this master. These addresses point to Configuration data base CRT records, which are described above under "**crt_record**". This array is indexed using the global VAR "**crtord**", which is declared in the INCLUDE module [TRVAR and described in Sections 9 and 11.3.

The last item in the "**master_record**" is the Version date/time stamp for the Configuration data base called "**confi_version**". Program INIT displays this value on the system console as part of the TRAMCON logo when the TRAMCON software is being booted up. Program CMMD displays this value on the command line in response to the VE command. This Time/Date is stored as the total number of seconds since midnight, 1 January 1970. This two-word integer value is unpacked by the routine "**DayTime**", which is defined in library $TRLIB and described in Section 8.2.4.1 of this manual.

```
links_record = ARRAY [0..max_links_per_net-1{250}] OF {1750 wds}
                 RECORD {7 wds}
                 site1, site2: site_record_ptr; {4 wds}
                 comm_equipment: equipment_record_ptr; {2 wds}
                 link_id: five_chars {1 wd}
                 END;
links_record_ptr = ^links_record;
```

The record type "**links_record**" is the Configuration Data Base record that defines the communications network monitored by the given master and is found in file (LINKS. Each TRAMCON master has one "**links_record**" defined in the data base. Each entry in this array describes a communications link at least partially monitored by this master. The locations of the two ends of the link are specified by "**site1**" and "**site2**". The communications equipment is identified by "**comm_equipment**". Since the communications equipment must be the same for both ends of any link, this field has been moved from the "**linkend_record**", where it is redundant information, to this record. The same holds true for the "**link_id**".

```
    net_segments = {2 wds}
      RECORD
      short_segment_name: dictionary_ptr; {1 wd}
      last_link_ptr: nill..max_links_per_net{250} - 1 {1 wd, -1 = undefined}
      END;
    net_masters = {3 wds}
      RECORD site_ptr:site_record_ptr; {2 wds}
      master_name_qualifier:CHAR {1 wd}
      END;
    network_record = {140 wds}
      RECORD
      segment_info:
        ARRAY [0..max_segments_per_net{25}-1] OF net_segments; {50 wds}
        {End of data is the first short_segment_name that is nil.}
      master_info: ARRAY [1..max_masters_per_net{30}]OF net_masters;{90 wds}
         {End of data is the first site_record_ptr that is nil.  The index
         into this array is also the DS node number of that master. Note
         that this array is not zero indexed because DS does not like a node
         number of zero.}
      END;
  network_record_ptr = ^network_record;
```

The record type "**network_record**" describes the Configuration Data Base network record found in file (NET. Each master has one "**network_record**" defined in the data base. This network record is used to describe two networks. The network of TRAMCON Segments composed of remote units is described in "**segment_info**". The network linking all TRAMCON masters is described in "**master_info**". The software system known as DS handles the communication on the master-to-master network. The DS node numbers are inferred from the position of a master in the array "**master_info**".

```
    current_link_status_record = {764 wds}
      RECORD
      current_2states: {432 wds}}
        PACKED ARRAY[link_2state_ordinal{288}] OF
          PACKED RECORD {2 wds}
              {machine word 1 (16-bits) }
          just_cleared, new_alarm: BOOLEAN;jday: nine_bits; hour: five_bits;
              {machine word 2 (16-bits) }
          alarm_set: BOOLEAN; tr_begin_end: 0..7; minute, second: six_bits
          tr_ord: INT {1}
        END; {current_2states}
    current_a2ds: ARRAY[a2d_ordinal{6}] OF INT; {6 wds}
    current_digitals: ARRAY[digital_ordinal{22}] OF INT; {22 wds}
    hist_a2d: ARRAY[a2d_ordinal{6}] OF hist_array; {96 wds}
    hist_digital: ARRAY[digital_ordinal{22}] OF hist_array; {352 wds}
    END; {current_link_status_record}
  current_alarm_ptr = ^current_link_status_record;
```

The current response for each CATEGORY of each remote unit defined on this master is stored in the HEAP variable "**remote_status[remoteord].cat_status**". This variable is of type "**current_link_status_record**" and contains the information specified above.

For each two-state value in a response, the time/date at which the alarm appeared or went away is stored in "**jday**", "**hour**", "**minute**", and "**second**". The value "**just_cleared**" indicates that the alarm was on in the last response, but is not on in the current response. The value "**new_alarm**" indicates that the alarm is on in the current response, but was not on in the previous response. The value "**alarm_set**" indicates that the alarm is ON in the current response. The values "**tr_begin_end**" and "**tr_ord**" are used to keep track of TRUNK (DIGROUP) alarms. If "**tr_ord**" has a value greater than nill then the given two-state is a TRUNK alarm that either begins or ends (value of "**tr_begin_end**") at the given linkend.

```
  parm_record =  {832 wds, record for disc file (CURVE }
    RECORD       {category_ordinal=-1..2, a2d_ordinal=0..5 }
    cal_curves: ARRAY[category_ordinal{4}] OF
                  ARRAY[a2d_ordinal{6}] OF hist_array; {384 wds}
    a2d_bottom , a2d_top , a2d_amber , a2d_red:
      ARRAY[category_ordinal{4},a2d_ordinal{6}] OF INT; {96 wds}
    digital_bottom , digital_top , digital_amber , digital_red:
      ARRAY[category_ordinal{4},digital_ordinal{22}] OF INT {352 wds}
    END;
```

Each "**parm_record**" has all the parameter calibration curves and threshold (analog and digital) for all categories for one remote. Along with curves and thresholds, the top and bottom range values for each parameter are stored in this record. Program INIT reads these values in from disc file (CURVE and places them in EMA record "**remote_status[remoteord]**".

```
  files_read = (archiv , calcurve);
  counted_array = PACKED ARRAY[link_2state_ordinal{144}]OF BOOLEAN;{9 wds}
  counts_array = ARRAY[0..max_counts_per_link{20}-1] OF {80 wds}
    PACKED RECORD {4 wds}
    val, yr: INT;  jdy: nine_bits; hr: seven_bits; minut, secs: byte
    END;
  cn_record = ARRAY[category_ordinal{4}] OF {356 wds, record for file (CN}
    RECORD {89 wds}
    cn_vals:counts_array;{80 wds}
    cn_counted:counted_array {9 wds}
    END;
```

The record type "**cn_record**" describes the records found in disc file (CN. These records contain the information for all the two-state values that are being counted. The TRAMCON operator can designate certain two-state alarms as alarms that the software should tally each time a response is received with the particular alarm set. The array "**cn_vals**" indicates which two-states have been so designated for each CATEGORY of each remote unit. Array "**cn_counted**" contains the actual counting information for each counted two-state. The actual count is stored in "**val**". The time/date at when the counting started is stored in "**yr, jdy, hr, minut, secs**".

```
     remote_status_record = {1210 wds}
       RECORD
       extent_of, next_extent: INT; {2 wds}
       no_answer , parity_err , bad_response , simulating: BOOLEAN; {4 wds}
       ss_alarms: ARRAY[-1..0, 1..2] OF link_2state_ordinal{144}; {4 wds}
       next_archive_record , parm_status: INT; {2 wds}
       parm_data: parm_record;{832 wds, initialized by INIT from file (CURVE}
       counts: cn_record;     {356 wds, initialized by INIT from file (CN }
       file_reader_cnt: ARRAY[files_read{2}] OF INT; {2 wds}
       cat_status: ARRAY[category_ordinal{4}] of current_alarm_ptr {8 wds}
       END;
     remote_status_ptr = ^remote_status_record;
```

The only reference to the two-word HEAP pointer "**remote_status_ptr**" is for the field "**remote_status**" in the record "**segment_status_record**" below. There is one two-word pointer for each possible remote unit. Currently, "**max_segments_per_master**" is set at 4 and "**max_remotes_per_segment**" is set at 21. Therefore, there are 84 two-word "**remote_status_ptr**" pointers allocated in the HEAP. These 84 pointers indicate the "**remote_status_records**" described above, each of which consumes 1210 words of HEAP. To conserve scarce HEAP space, pointers were used rather than allocating space for a "**remote_status_record**" for each possible remote unit which would require 1210 x 84, or 101,640 words. With the pointers, space for a "**remote_status_record**" is allocated only for the remote units that are defined on the given master. Unused pointers are set to NIL. This use of EMA pointers adds one level of indirection when addressing the remote unit data and adds the 160 words of pointer data as overhead. The extra address processing is NOT noticeable, and the 160 words extra is more than offset by the 1210 word savings if even one remote unit of the possible 84 is not defined.

```
     segment_status_record = {268 wds}
       RECORD
       nbr_remotes , previous_remotes: INT; {2 wds}
       remote_status:
         ARRAY[segment_remote_ordinal{21}] OF remote_status_ptr; {42 wds}
       pcm_counts: pcm_histogram_array; {200 wds}
       main_resp,tout,wait_ext,arch_it,al_update,b_r,not_ans,NA,p_e,resp,
       extended,time_res,time_pro,time_dis,time_tra,time_disp:BOOLEAN;{16 wd}
       disc_start,disp_start,poll_timer: INTEGER; {6 wds}
       currently_polled: set_of_remotes {2 wds}
       END;
```

The dynamic information for each Segment monitored by a given master is stored in the HEAP variable "**heap^.segment_status**", which is of type "**segment_status_record**". The number of remote units defined for each segment is stored in "**nbr_remotes**" by INIT. Program INIT also computes the number of remote units defined before each segment and places this value in "**previous_remotes**". This "**previous_remotes**" value is used through the On-Line software to calculate record positions in the disc files for given remote units.

The dynamic status of each remote unit for each segment is pointed to by the array "**remote_status**". The indicators "**main_resp**", "**tout**", "**wait_ext**", "**b_r**", "**not_ans**", "**NA**", "**p_e**", "**resp**", **and** "**extended**" are used by the response processing routines to coordinate the processing of multiple remote units. The indicators "**time_res**", "**time_pro**", "**time_dis**", "**time_tra**", "**time_disp**", "**disc_start**", "**disp_start**", **and** "**poll_timer**" are used by the response processing routines to collect statistics concerning the time involved in various stages of the response processing operation.

heap_ptrs = {4335 wds}
RECORD

The record "**heap_ptrs**" contains the top-level pointers to all the TRAMCON configuration data stored in EMA and all the run-time data such as current status information for each link end of each remote unit of each segment currently being monitored. This record is set up by the program INIT and communicated to other programs through the class number "**heap_class**" by passing it to a program as run-string parameter "**parms[1]**" each time the program is scheduled (refer to Section 4.3). INIT does a class write of "heap^" with the save bit on. Any program wanting access to the EMA data must make a two-word CLASS GET on the class number "**heap_class**". For most programs, this CLASS GET is done by the routine "**allocate_EMA**". The first word address of the HEAP is placed into the global VAR "**heap**", which is of TYPE "**heap_ptrs**" and declared in the VAR INCLUDE module [TRVAR. All the following definitions are fields within the RECORD TYPE "**heap_ptrs**".

software_date: INTEGER; {2 wds}
software_version: REAL; {2 wds}   Referenced by CMMD

The time/date-stamp and version number for the TRAMCON On-Line software are read from disc file (DATE and stored here by program INIT. Program INIT displays these values on the system console as part of the TRAMCON logo when the TRAMCON software is being booted up. Program CMMD displays these values on the command line in response to the VE command. The time/date is stored as the total number of seconds since midnight 1 January 1970. This two-word integer value is unpacked by the routine "**DayTime**", which is defined in library $TRLIB and described in Section 8.2.4.1 of this manual. The software version number is stored as a floating point number with one decimal point and displayed as is. Version numbering schemes that incorporate more than one decimal point in the version numbers, such as 1.8.1, will not be properly stored or displayed by the present TRAMCON software. The example 1.8.1 can only be stored as the REAL number 1.81 and displayed the same way.

master: master_record_ptr; {2 wds} Referenced by All TRAMCON programs

All of the static Configuration data in EMA (except the dictionary, network, and links records) are accessed through this two-word EMA address. This address is technically the pointer to the master record, which contains the information that distinguishes one TRAMCON master computer from any other. Each TRAMCON master has one master record and that record is described above under "**master_record**".

network: network_record_ptr; {2 wds}

The two word EMA address "**network^**" points to the TRAMCON master computer network record, which is used by the interprocessor software to determine the master computer network connectivity.

links: links_record_ptr; {2 wds, Ref by INIT, MA}

Closely related to the network record is the "**links**" record. Each entry in this array contains site record pointers for all communication links defined on the entire TRAMCON network. This information is used by the program INIT in subroutine SCALE to set up the information in "**heap^.latlons**" (see below).

dictionary: dictionary_record_ptr; {2 wds}
Referenced by AL,CC,CMMD,CN,CR,ED,HI,INIT,KYBRD,LO,LS,MA,ME,MS,
PA,PC,PF,PH,SE,SS,SW,TH,US via routine "read_dict" in
library $TRLIB (see Section 8.2.4.1).

The Configuration data base dictionary, pointed to by "**dictionary**", is constructed by the Configurator program automatically, as the other types of data are entered (see Configurator Manual). The dictionary is one long array of characters and is referenced by the array index of the first character of the particular dictionary WORD ("**dictionary_word**" defined above). Any datum having the type "**dictionary_ptr**" points into the dictionary in this manner. Each word is terminated by the ASCII character delete (octal 177) and all TRAMCON software modules use the routine "**read_dict**" in library $TRLIB (see Section 8.2.4.1.) to retrieve a word from the dictionary. This, of course, implies that the ASCII delete character is not a valid character in a dictionary word. The only direct references to the dictionary pointer are made by the subroutine "**read_dict**" and by the program INIT (see Section 4.1).

segment_status: ARRAY[master_segment_ordinal{4}] OF
segment_status_record;{1072 wds} Referenced by CMMD,INIT,MTRP,PLRP

In addition to the static segment information stored in the segment records discussed above, run-time data is kept and pointed to by "**heap^.segment_status[segord]**". Each element of this array is a "**segment_status_record**", which contains the following information for each active segment. The number of logical remote units defined for this segment is kept in "**nbr_remotes**". "**Previous_remotes**" is an accumulation of "**nbr_remotes**" for all segments preceding this one in the data definition. In other words, if segment DEB2A is defined in position 0 in the array "**heap^.master^.segment**", then its "**nbr_remotes**" still equals 7, and its "**previous_remotes**" equals 0. If segment FKT-N1 is defined second, then its "**previous_remotes**" equals 7. If segment DEB1 is defined third, then its "**previous_remotes**" equals 20 (7 DEB2A remotes + 13 FKT-N1 remotes). It is important to understand these two values because they are used to address the proper run-time memory information as well as to compute the correct disc record number for a given remote unit in such files as (ARCH or (HIST.

For each link end defined, a snapshot of the status of that link end is kept in EMA and pointed to by "**current_alarm_ptrs**". Each time a response is received from an active remote, either program PLRP or program MTRP processes the response and compares it with the information kept in "**current_alarm_ptrs**". If any change is found, the new response replaces the old data in "**current_alarm_ptrs**". Since "**current_alarm_ptrs**" records are allocated only for link ends that are defined to conserve EMA storage area, the records are not placed in the "**heap^**" record itself. Instead, pointers are kept for all possible link ends, defined and undefined, in "**heap^**" with the pointers corresponding to undefined (unused) linkends set to -1 (nil). This way, a simple scheme can be used to reference the records and still realize the most efficient storage method.

> **stack_alloc: PACKED ARRAY [0..stack_alloc_size{50}-1] OF byte;{25 wds}**
> **next_id: byte; {1 wd, id number used for program ident in stack_alloc}**
> **Referenced by All TRAMCON programs through routine "Allocate_EMA" in library $TRLIB (see Section 8.2.4.1)**

The EMA allocation routine "**Allocate_EMA**", described in Section 8.2.4.1, uses the variables "**heap^.stack_alloc**" and "**next_id**" to dynamically allocate stack space in EMA for each program. Programs require stack space for copies of recursive routines or storage of HEAP2 parameters. The best statement we can make about stack usage is that it is very poorly understood by us; therefore, overt use of the stack has been kept to a minimum. The two programs PLRP and MTRP use a recursive routine called "**expression_tree**" and for that reason they request a few hundred words of stack space.

> **max_crt, max_segment: INT; {2 wds}**

The two values "**max_crt, max_segment**" are set by program INIT and are used throughout the On-Line software to terminate loops that search through all defined CRTs and all Segments. They serve make these searches more efficient by allowing the loops to terminate without searching all possible entries.

```
real_kybrd_class, {Ref by KYBRD or any program wishing to
                   issue a READ request for a given terminal}
cmmd_class,       {Ref by CMMD, KYBRD or any program wishing
                   to programmatically enter a TRAMCON command such as
                   scheduling the Default Display}
plrp_class,       {Ref by PLRP, PM, POLL}
mtrp_class,       {Ref by MTRP, PM, POLL}
poll_class,       {Ref by POLL, PLRP, MTRP, SW}
si_class,         {Ref by SI, routine "simple_cmd" in CMMD}
logon_class,      {Ref by LON}
msg_class,        {Ref by MSG or any program wishing to display a msg}
```

```
      lgoff_class,      {Ref by LOF}
      di_segrem: INT;   {Ref by CMMD (simple_cmd), PLRP, MTRP thru
                         routine "print_response" in $MPLIB}
```

Communication between programs and most TRAMCON I/O is performed using the HP software feature called "class" IO. All the CLASS numbers are allocated and stored in EMA by the program CMMD. For example, if a program wishes to communicate with the program CMMD, it attaches its message to the class number "**cmmd_class**" and CMMD will eventually read and process the message.

```
      current_crt: ARRAY[master_crt_ordinal{5}] OF {2340 wds}
        RECORD {468 wds}
        crt_down, color_crt, graphic_crt, graphics_mode,
          insert_cmd_line, print_dsp: BOOLEAN; {6 wds}
        whole_x, whole_y, half_x, half_y: INT; {4 wds, Graphics boundaries}
        latlons: ARRAY[master_segment_ordinal{4}] OF {168 wds}
          RECORD
          lats,lons: ARRAY[segment_remote_ordinal{21}] OF INT {42 wds}
          END;
        up_class , crt_rn: INT; {2 wds}
        crt_class: INT;      {class number to read crt }
        crt_recnbr: INT;     {record number on file (CRT:TR )
        crt_eqt: two_chars;  {crt RTE equipment number (ASCII) set by INIT}
        current_msgord: INT; {negative if msg currently displayed, 0 if no
                              current msg, positive if msg exists, but not
                              displayed.}
        msg_ords , msg_segords , msg_remoteords , msg_priorities,
          msg_lengths: ARRAY[crt_msg_ordinal{5}] OF INT; {25 wds}
        msgs: ARRAY[crt_msg_ordinal{5}] OF sixty_chars; {150 wds}
        current_display, default_display, crtlu_alfa: two_chars; {3 wds}
        current_segord, fkey_entry: INT; {2 wds}
        operator_name: sixteen_chars; {8 wds}
        remotes_displayed, alarms_acknowledged,
          remotes_to_print , alarms_inhibited: remotes_array; {32 wds}
        old_cursor: RECORD x,y: INT END; {2 wds}
        last_cursor: twenty_chars; {10 wds}
        first_line, lines: ARRAY[1..15] OF INT; {30 wds}
        sav_dsp, old_dsp: two_chars; {2 wds}
        kybrd_class, line_nbr, nbr_lines, pgs_remaining, cur_page,
          prev_pages, misc1, misc2, misc3, misc4, misc5, misc6,
          max_dsp_ln, max_key, locked_ln, sav_fkey_entry, max_page,
          sav_len, sav_echo, sav_binary: INT {20 wds}
        END;
      Referenced by all display programs and CMMD
```

Information describing the current state of each terminal defined on the given master is kept in HEAP array "**heap^.current_crt**". The array is indexed by the global VAR "**crtord**", which has the range defined by the TYPE "**master_crt_ordinal**" above and with ordinal 0 reserved for the system console. The flag "**crt_down**" is true if the software is unable to communicate with the given terminal. The variable "**color_crt**" is true if the terminal is capable of displaying colors and is set by LO when logging ON

Section (8.2). The flag "**graphic_crt**" is also set by LO if the terminal is capable of performing graphics functions such as vector drawing. The variable "**insert_cmd_line**" is used by various programs to coordinate the display of the command entry line on a screen that may or may not be displaying data on line 22. For instance, if a screen currently shows 25 lines of alarm/status data, "**insert_cmd_line**" should be true so that if the operator wishes to enter a command, or a message needs to be shown on line 22, the software will insert a line but not overwrite the data in line 22.

"**Print_dsp**" is set to true to inform a display program to route its output to the printer and not to the screen. The global integers "**whole_x**", "**whole_y**", "**half_x**", **and** "**half_y**" are the graphics dimensions for the given terminal and are used by any program (e.g., MA) wishing to produce graphics information on the screen. "**whole_x**" and " **whole_y**" are set to the maximum values for the horizontal and vertical graphics screen dimensions respectively. All four values are set by the program LO once the terminal type has been determined. The latitude and longitude are adjusted to screen graphics coordinates and placed into array "**latlons**" by the program INIT at bootup so that the program MA can read the data from this array to produce the segment map.

time_it: ARRAY[master_segment_ordinal{4}, 1..5] OF BOOLEAN; {20 wds}
time_val: ARRAY[master_segment_ordinal{4}, 1..5] OF INTEGER; {40 wds}

Referenced by US, MTRP, PLRP

The HEAP arrays "**time_it**" and "**time_val**" are used to analyze the response processing and disc access functions of the TRAMCON software. The program US is used to both designate which processes are to be timed and to display the timing values.

Program US informs the PLRP and MTRP programs about which processes to time for which segments by setting the elements in "**time_it**" to true. Program US is also used to reset the timing values. There are currently five processes set up to be timed. All five are related to the processing of a response from a remote unit. The first is the overall response processing time, which starts just after a poll message is sent and ends in routine "**update_displays**" after all display screens have been updated for the given response. The other four are subsets of the first. The second value represents the CPU time processing the response. The time begins when routine "**process_response**" (see Section 8.2.4.2) is entered and ends when "**process_response**" is exited. The third value represents the time spent accessing the disc and is primarily set by routine "**archive_it**" (see Section 8.2.4.2). The fourth value is the time elapsed while the response is being transmitted and should roughly be the length of the response in bits divided by 300 bps. The last value is the time required to update all displays and begins and ends in routine "**update_displays**" (see Section 8.2.4.2).

EMA_start , EMA_end ,
EMA_required: ARRAY[master_segment_ordinal{4}] OF INTEGER;{24 wds}
Referenced by INIT, SE

All three of these values are set by the program INIT at bootup. For each segment defined in the given master's data base, "**EMA_start**" is set to the two-word EMA address at which the dynamic data for that segment starts and "**EMA_End**" is set to the last address. "**EMA_required**" represents the number of machine words needed for these dynamic data and is simply the difference between "**EMA_start**" and "**EMA_End**". For diagnostic purposes, program SE will display this information. These data were intended to support the activation/deactivation of segments. These values can be used to decide whether more than two segments could be monitored by a TRAMCON master.

statz: statz_record; {650 wds, TRAMCON performance statistics}

Referenced by CMMD,CF,HR,INIT,US

Statistical information on the performance of the transmission system and operator usage of the TRAMCON system is kept in record "**statz**" and recorded permanently on disc file (STATZ once each hour, on the hour, by the program HR. This information can be viewed and/or initialized by the operator using the command "**US**". The record currently contains a count of each command "**cnt_cmds[cmd,crtord]**" that has been entered at each CRT. These values are tallied by the program CMMD as each command is received from a keyboard. The array "**transmission[segord][remoteord,msg_status]**" contains statistics on the overall transmission status of each response received from each remote unit. The status of each response falls into one of the categories that are enumerated in the type specification "**msg_status**" defined above. Any future additions to the statistics gathering function should be added to this record and the disc file (STATZ, and the proper changes made to the programs US and CF with recompilation required for programs INIT and HR.

archive_idx: archive_record; {125 wds}

Referenced by AL, PLRP, MTRP

The record "**archive_idx**" is a memory copy of the first record in the disc file (ARCH. This copy is kept in memory so that the index record does not have to be rewritten to disc each time a new archive record is created, thus reducing disc access time required to process a response. To ensure that the disc copy is fairly current, the program HR updates the first record in file (ARCH from this copy on the hour.

diag, access_restricted: BOOLEAN; {2 wds}

Referenced by CMMD (simple_cmd) and by any program that has code to be triggered by the "d" option in a TRAMCON command

The "**diag**" flag is a toggle flag. That is, each time the "**d**" option appears in any TRAMCON command the value of "**diag**" is set to NOT "**diag**". This flag is not used if the master password is not entered (i.e., "**restricted_access**" is true). Any program can have hidden code that executes only if the "**diag**" flag is true (refer to Section 12.1).

heap^.access_restricted - Referenced by CMMD

The "**access_restricted**" flag is set and cleared by program CMMD in response to the "**pw,-1**" command entry. This flag is described in detail in Section 12.3.

```
        resp_stats: ARRAY [master_segment_ordinal{4}] OF {12 wds}
                    PACKED RECORD {3 wds}
                    remote_num: INT; {1 wd}
                    cos,fil: BOOLEAN; jday: nine_bits; hr: five_bits; {1 wd}
                    fil2: 0..15;{4 bits} min, sec: six_bits {1 wd}
                    END
        Referenced by PLRP, MTRP
```

The HEAP array "**resp_stats**" contains information on the last response received for each segment defined in the data base. This information is placed here by the programs PLRP and MTRP via routine "**process_response**" and is ignored by any TRAMCON programs.

To access any of the HEAP data described above, a TRAMCON On-Line program must declare one two-word EMA pointer and set that variable to the first-word-address (FWA) of the HEAP. The first variable in the INCLUDE global VAR module [TRVAR is called "**heap**" and is just such a pointer or FWA. The TYPE of the variable "**heap**" is "**heap_ptr**", which is defined to be a pointer to a record variable of the TYPE "**heap_ptrs**" just described.

```
      heap_ptr = ^heap_ptrs;
```

There is only one VAR in the TRAMCON source code that is declared as TYPE "**heap_ptr**". The VAR "**heap**" is declared in the global VAR INCLUDE module [TRVAR. Each TRAMCON On-Line program that accesses the shared data described above uses the global VAR "**heap**" as the FWA of the Pascal HEAP or shared data area called EMA. Since a "**heap_ptr**" points to "**heap_ptrs**", the value of "**heap**" is the two-word address of the first field "**network**" in the RECORD "**heap_ptrs**" described above. There is also only one "**heap_ptrs**" record allocated in the HEAP (EMA) area. This FWA is established by the program INIT when the TRAMCON On-Line software is booted up. Program INIT places the newly acquired HEAP FWA into a CLASS output buffer that can be read indefinitely and will remain for the life of the TRAMCON software. Any TRAMCON programs wishing to access the HEAP data can get the CLASS number associated with the buffer containing the FWA of the HEAP by either calling the routine "**get_parms**" immediately after activation or reading the value from the disc file (DATE. The CLASS number associated with the buffer containing the two-word FWA is passed by program CMMD to any program it schedules. Any program scheduled by CMMD first calls "**get_parms**", which sets the value of the global VAR "**parms[1]**" to the desired CLASS number and then calls the routine "**allocate_EMA**", which in turn reads the FWA of the HEAP using the CLASS number it found in "**parms[1]**". For further detail, refer to the "**allocate_EMA**" description in Section 8.2.4.1. A few programs, such as the DT support program ARPTR, wish to access the HEAP data but are not scheduled by program CMMD and, therefore, are not passed the CLASS number that allows them to acquire the FWA of the HEAP. For these programs, a duplicate of the CLASS number that leads to the FWA of the HEAP is placed in the field "**heap_class_no**" in the one record in the disc file (DATE. The contents of the record stored in file (DATE are discussed previously in this section under "**date_record**".

11.2 Shared Data (EMA)

Most of the data used by the TRAMCON programs reside in a partition of main memory called SHAR1. This is a type of memory known as Extended Memory Area (EMA), which can be shared by any number of active programs at any given time. Program that wants to access these data must include the code block [TRVAR (10.3), which has a declaration of a variable called "**heap**", or the program must explicitly declare the GLOBAL variable "**heap**". Either of the above must be done immediately following the inclusion of the definitions block [RECR3 (10.1) with no preceding CONST or VAR declarations.

The first action in the program body should be a call to routine "**get_parms**", which gets the class number "**heap_class**" as the value of "**parm[1]**". Next, the routine "**allocate_EMA**" (7.2.4.1) is called to set up the pointer "**heap**", which points to all this shared information.

The connection between the correct partition of memory and the executable code that references these data is made at load (or link) time by directing the linker with the statement "**sh,sharl**" (7.2.5). As the following description shows, the identifier "**heap**" is the master pointer to all the shared data and once the above procedures are performed, any of the shared data listed below can be accessed through this pointer.

The program INIT physically allocates all the EMA at bootup and initializes the static portion of this shared memory with information from the configuration data files on disc. The EMA addresses for each segment's dynamic data are recorded by INIT in the EMA values EMA_start, EMA_end, and EMA_required. These addresses were to be used by the TRAMCON software to implement the activation or deactivation of segments.

Figure 55 is a list of the EMA identifiers and their corresponding data types. Any program wishing to reference these data, having followed the above procedures, must use the identifiers listed in Figure 55. Also, most TRAMCON programs make extensive use of the Pascal WITH statement. Therefore, care must be taken in looking through the code to determine the complete identifier involved in any given reference. Coordination with the list in Figure 55 is recommended. The EMA identifiers in Figure 55 are shown as they would appear in the software after applying the following rules:

(1) All identifiers with a type ending in "_ptr" are actual PASCAL pointers and must be followed by ^ to reference the actual data. The only exception is the type "**dictionary_ptr**", which is simply an index into the character array "**dictionary^**". For example, to reference a segment record the identifier would be:

```
heap^.master^.segment[segord].seg_ptr^
```

```
    Identifier                                       Type
heap                                              heap_ptr
  software_date                                   INTEGER
  software_version                                REAL
  master                                          master_record_ptr
    site_ptr                                      site_record_ptr
      site_code                                   dictionary_ptr
      site_name                                   dictionary_ptr
      master_flag                                 BOOLEAN
      country                                     dictionary_ptr
      latitude                                    REAL
      longitude                                   REAL
      service_branch                              dictionary_ptr
    master_name_qualifier                         CHAR
    segment[segord]
      seg_ptr                                     segment_record_ptr
        short_segment_name                        dictionary_ptr
        long_segment_name                         dictionary_ptr
        currently_polled                          set_of_remotes
        remote_info[remoteord]                    remote_record_ptr
          remote_polling_id                       byte
          site                                    site_record_ptr
            (see "heap^.master^.site_ptr above)
          site_equipment                          equipment_record_ptr
            equipment_name                        dictionary_ptr
            two_states[2stord]                    two_state_record
              alarm_name[wordord]                 dictionary_ptr
              alarm_type                          nibble
              special_name_flag                   BOOLEAN
              pcm_port                            byte
```

Figure 55. List of HEAP (EMA) identifiers.

```
             Identifier                    Type
          equip_a2d[a2dord]           parameter_record
            param_name[wordord]       dictionary_ptr
            param_type                byte
            param_units               dictionary_ptr
          equip_digital[digord]       parameter_record
            (see equip_a2d above)
          relays[relayord]            relay_record
            relay_name[wordord]       dictionary_ptr
            relay_type                byte
            relay_status              INT
            open_name                 dictionary_ptr
            closed_name               dictionary_ptr
          combos[comboord]            combo_record
            combo_name[wordord]       dictionary_ptr
            combo_type                BOOLEAN
            expression[node]          expression_tree
              op                      INT
              left_link               INT
              right_link              INT
            link_info[linkord]            link_record_ptr
          specific_name[x,y]          specific_name_record
            name[wordord]             dictionary_ptr
            alarm_number              -1..max_2states_per_link-1
          relay_exists[x,relayord]    PACKED ARRAY[0..1,relayord]OF BOOLEAN
        remote_equip_type             remote_types
        remote_equip_name             dictionary_ptr
      trunk_info[trunkord]            trunk_record_ptr
        trunk_id                      dictionary_ptr
        links_in_trunk[node]          trunk_link_array
          trunk_links                 link_def_ptr
          trunk_port                  byte
    segment_lu                        INT
    poll_monitor                      0..2
    alternate_masters[masterord]      DS_node
  crt_ptr[crtord]                     crt_record_ptr
    comm_info                         comm_info_record
      baud                            byte
      auto_answer                     BOOLEAN
      modem                           BOOLEAN
    terminal_type                     byte
    printer_type                      0..7
    location                          site_record_ptr
      (see "heap^.master^.site_ptr" above)
    location_qualifier                CHAR
  confi_version                       INTEGER
network                               network_record_ptr
  segment_info[segord]                net_segments
    short_segment_name                dictionary_ptr
    last_link_ptr                     -1..max_links_per_net-1
```

Figure 55. (cont.)

```
        Identifier                         Type
     master_info[masterord]              net_masters
       site_ptr                          site_record_ptr
         (see "heap^.master^.site_ptr" above)
       master_name_qualifier             CHAR
   links                                 links_record_ptr
   dictionary                            dictionary_record_ptr
   segment_status[segord]                segment_status_record
     nbr_remotes                         INT
     previous_remotes                    INT
     remote_status[remoteord]            remote_status_ptr
       extent_of                         INT
       next_extent                       INT
       no_answer                         BOOLEAN
       parity_err                        BOOLEAN
       bad_response                      BOOLEAN
       simulating                        BOOLEAN
       ss_alarms[-1..0, 1..2]            link_2state_ordinal
       next_archive_record               INT
       parm_status                       INT
       parm_data                         parm_record (832 wds, file (CURVE )
         cal_curves[category, a2dord]    hist_array
         a2d_bottom[category, a2dord]    INT
         a2d_top                         INT
         a2d_amber                       INT
         a2d_red                         INT
         digital_bottom[category,digord] INT
         digital_top[category, digord]   INT
         digital_amber[category, digord] INT
         digital_red[category, digord]   INT
       counts[category]                  RECORD
         cn_vals[0..max_counts_per_link] counts_array
           val                           INT
           yr                            INT
           jdy                           nine_bits
           hr                            seven_bits
           minut                         byte
           secs                          byte
         cn_counted[link_2state_ordinal] counted_array {PACKED BOOLEAN}
       file_reader_cnt                   ARRAY[files_read] OF INT
       cat_status[category]              current_alarm_ptr
         current_2states[link_2stateord] PACKED RECORD
           just_cleared                  BOOLEAN
           new_alarm                     BOOLEAN
           jday                          nine_bits
           hour                          five_bits
           alarm_set                     BOOLEAN
           fill                          0..7
           minute                        six_bits
           second                        six_bits
         current_a2ds[a2dord]            INT
```

Figure 55. (cont.)

Identifier	Type
current_digitals[digord]	INT
hist_a2d[a2dord]	hist_array
hist_digital[digord]	hist_array
pcm_counts[1..2, trunkord]	INT
main_resp	BOOLEAN
tout	BOOLEAN
wait_ext	BOOLEAN
arch_it	BOOLEAN
al_update	BOOLEAN
b_r	BOOLEAN
not_ans	BOOLEAN
NA	BOOLEAN
p_e	BOOLEAN
resp	BOOLEAN
extended	BOOLEAN
time_res	BOOLEAN
time_pro	BOOLEAN
time_dis	BOOLEAN
time_tra	BOOLEAN
time_disp	BOOLEAN
disc_start	INTEGER
disp_start	INTEGER
poll_timer	INTEGER
currently_polled	set_of_remotes
stack_alloc[x]	byte
next_id	byte
max_crt	INT
max_segment	INT
real_kybrd_class	INT
cmmd_class	INT
plrp_class	INT
mtrp_class	INT
poll_class	INT
si_class	INT
logon_class	INT
msg_class	INT
lgoff_class	INT
di_segrem	INT
current_crt[crtord]	current_crt_record
crt_down	BOOLEAN
color_crt	BOOLEAN
graphic_crt	BOOLEAN
graphics_mode	BOOLEAN
insert_cmd_line	BOOLEAN
print_dsp	BOOLEAN
whole_x	INT
whole_y	INT
half_x	INT
half_y	INT

Figure 55. (cont.)

Identifier	Type
latlons[master_segment_ordinal]	RECORD
lats[remoteord]	INT
lons[remoteord]	INT
up_class	INT
crt_rn	INT
crt_class	INT
crt_recnbr	INT
crt_eqt	INT
current_msgord	INT
msg_ords[crt_msg_ordinal]	INT
msg_segords[crt_msg_ordinal]	INT
msg_remoteords[crt_msg_ordinal]	INT
msg_priorities[crt_msg_ordinal]	INT
msg_lengths[crt_msg_ordinal]	INT
msgs[crt_msg_ordinal]	sixty_chars
current_display	two_chars
default_display	two_chars
crtlu_alfa	two_chars
current_segord	INT
fkey_entry	INT
operator_name	sixteen_chars
remotes_displayed	remotes_array
alarms_acknowledged	remotes_array
remotes_to_print	remotes_array
alarms_inhibited	remotes_array
old_cursor	RECORD
x	INT
y	INT
last_cursor	twenty_chars
first_line	ARRAY[1..15] OF INT
lines	ARRAY[1..15] OF INT
sav_dsp	two_chars
old_dsp	two_chars
kybrd_class	INT
line_nbr	INT
nbr_lines	INT
pgs_remaining	INT
cur_page	INT
prev_pages	INT
misc1	INT
misc2	INT
misc3	INT
misc4	INT
misc5	INT
misc6	INT
max_dsp_ln	INT
max_key	INT
locked_ln	INT
sav_fkey_entry	INT
max_page	INT

Figure 55. (cont.)

Identifier	Type
sav_len	INT
sav_echo	INT
sav_binary	INT
time_it[segord, 1..5]	BOOLEAN
time_val[segord, 1..5]	INTEGER
EMA_start[segord]	INTEGER
EMA_end[segord]	INTEGER
EMA_required[segord]	INTEGER
statz	statz_record
cnt_cmds[un..us, crtord]	INT
transmission[segord][remoteord, msg_status]	INT
archive_idx	archive_record
diag	BOOLEAN
access_restricted	BOOLEAN
resp_stats[segord]	PACKED RECORD
remote_num	INT
cos	BOOLEAN
fil	BOOLEAN
jday	nine_bits
hr	five_bits
fil2	0..15
min	six_bits
sec	six_bits
heap_ptrs_size	INT
master_rec_size	INT
network_rec_size	INT
links_rec_size	INT
segment_rec_size	INT
remote_rec_size	INT
linkend_rec_size	INT
equip_rec_size	INT
crt_rec_size	INT
trunk_rec_size	INT
site_rec_size	INT
cat_status_size	INT
rem_stat_size	INT
nbr_segments	INT
nbr_remotes	INT
nbr_linkends	INT
nbr_equipments	INT
nbr_crts	INT
nbr_trunks	INT
nbr_sites	INT
nbr_cat_status	INT
nbr_rem_status	INT

Figure 55. (cont.)

(2) The following short names for array subscripts are used. Their long definitions can be found in the description of [RECR3 in Section 11.1:

masterord = 0..max_masters_per_segment - 1
segord = master_segment_ordinal
remoteord = segment_remote_ordinal
linkord = 0..max_linkends_per_remote - 1
2stord = link_2state_ordinal
category = category_ordinal
a2dord = a2d_ordinal
digord = digital_ordinal
crtord = master_crt_ordinal
wordord = 0..max_words - 1
comboord = 0..max_combos_per_link - 1
relayord = 0..max_relays_per_link - 1
trunkord = 0..max_trunks_per_trunk - 1
node = 0..max_sites_per_trunk - 1

(3) Any type definitions used by the TRAMCON software that are NOT basic PASCAL types can be found in the description of [RECR3 in Section 11.1.

11.3 Global Data Definitions - [TRVAR, [MPVAR, and [DTVAR

The term "**GLOBAL**" as used in this section refers to the program-wide scope of the variables declared in these three INCLUDE modules and should not be confused with the even greater scope of shared data variables that are stored in the HEAP and are shared between programs. The global data declarations shown in the figures in this section are Pascal VAR declarations and are incorporated into any program module using the INCLUDE feature. That is, these VAR declarations are merged into the program SOURCE before compilation. These INCLUDE code blocks are referenced in a particular program module by placing an INCLUDE compiler directive, such as "**$INCLUDE '[TRVAR'$**", in the source code at the exact point where the VAR declaration is intended to go.

When the compiler encounters the INCLUDE directive, it replaces the directive statement with the actual VAR declaration that it finds on disc, using what is enclosed in single quote marks as the file name. The naming convention for INCLUDE source module files requires that the first character of the name be "[". For example, upon encountering the directive **$INCLUDE '[TRVAR'$**, the compiler opens the disc file named [TRVAR then places the contents into the source module being compiled and resumes compilation with the first line that was just inserted.

The advantage that the INCLUDE feature offers is that similar source code does not have to be repeated by hand for each module that uses those lines of code. Also, if any part of that code block is changed, all modules that include it are affected by simply recompiling. In this case, the code block is a VAR section. Any module that includes this block gets this exact set of variables made available for use throughout the module. These modules must be defined as global to the entire program module, that is, at lexic level 0,

and are always included as the first VAR definition. They must also be preceded by the INCLUDE module [RECR3, which contains the TYPE and CONST definitions used by many of the variables in these INCLUDE modules. A typical program module contains the following two directives immediately after the program statement: **$INCLUDE '[RECR3'$**
$INCLUDE '[TRVAR'$

NOTE

This set of variables is used by virtually all TRAMCON programs and by most of the routines in the two TRAMCON libraries $MPLIB and $TRLIB. Since the libraries are compiled separately from the program modules, it is extremely important that these definitions be included in all modules in exactly the same order and place so that the compiler gives the same relative address for corresponding variables. The order of the variables within each INCLUDE block is also very important. For example, the block [TRVAR is included in the library module $TRLIB and in the program module CMMD. There are references to the variable "crtord" in both the program CMMD and by several routines in $TRLIB that are in turn referenced by program CMMD. The program logic expects these references to be to the same physical memory location when the program CMMD is executing, but at the time of compilation, the compiler was not aware of both CMMD and the routines in $TRLIB since they were compiled separately. By placing the [TRVAR VAR definition in the same place in each module and by using the INCLUDE feature to ensure the exact same definition, the programer forces the compiler to generate the same addresses for the same identifiers.

Figure 56 shows the main global variable INCLUDE module [TRVAR that is used by almost all TRAMCON software modules. If any part of this module is changed, all the TRAMCON software must be recompiled and reloaded.

```
VAR heap: heap_ptr; parms: parm_array; id: byte;
    word, sname: dictionary_word;  soft_key, color: CHAR;
    i, j, k, crtord, segord, remoteord, category, linkord, sname_len,
      local_end,rnrq_status,crt_type,trunkord,last_ln,first_ln,ln: INT;
    time_alfa: time_str; print_it, refresh, colored: BOOLEAN;
    oppo_site, siteptr: site_record_ptr; equip: equipment_record_ptr;
    crtptr: crt_record_ptr;    rem_status_ptr: remote_status_ptr;
    remptr: remote_record_ptr; segptr: segment_record_ptr;
    cat_status_ptr: current_alarm_ptr; linksptr: link_def_ptr;
    linkendptr: linkend_record_ptr; trunkptr: trunk_record_ptr;
    crt_buff: cmd_str;  crt_IO_len: INT;
    $LINESIZE 1920$ outunit: TEXT; $LINESIZE 128$
```

Figure 56. Global data declaration module [TRVAR.

The following is a brief discussion of each variable declared in the INCLUDE module [TRVAR.

"heap" - The type, "heap_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual. This two-word integer is the first-word address of the type 2 HEAP that contains all the static configuration data and dynamic run-time data that is shared by most TRAMCON programs. As each program begins to run, this address is acquired by calling the routine "allocate_EMA". Refer to Section 8.2.4.1 for a complete description of the routine "allocate_EMA" and how this address is set. Refer to Section 11.2 for a discussion about the data stored in the HEAP.

"parms" - The type, "parm_array", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual. This array consists of five one-word integers and is used by all TRAMCON programs to store the five run-string parameters that are passed to each program as it is scheduled. The first value, "parms[1]", always contains the class number that allows the newly-scheduled program to acquire the first word address of the HEAP, the variable "heap" described above.

"id" - The type, "byte", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual. This single-byte value is used to identify any stack space the given program might have assigned to it. At this point, only the two programs MTRP and PLRP ask for stack space, and since these programs run continuously, this space is never returned. Refer to the Pascal/1000 Reference Manual, p. 8-17.

"word, sname" - The type, "dictionary_word", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. These two 30-character strings are used by TRAMCON programs to build entries from the Configuration data base dictionary. These dictionary words are built by the routine "read_dict", which is discussed in Section 8.2.4.1 of this manual. The variable "word" is used by most programs. "sname" is also used by the few programs that need two values from the dictionary at once.

"soft_key" - The type, CHAR, is a basic Pascal type. This byte of data is used by all programs that accept function keypresses from the keyboard. The routine "keypress" places single key presses in this value and is discussed in Section 8.2.4.1.

"color" - The type, CHAR, is a basic Pascal type. This byte of data is used as the ASCII representation of an integer value representing the color selection for escape sequences that are output to the terminal to alter the color being displayed. The possible values of "color" are defined as constants in the INCLUDE module [RECR3 and are discussed in Section 11.1 of this manual.

"i,j,k" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. These three one-word integer values are used by most programs as utility variables such as loop or array indices. Seldom are more than three of these variables needed. If more are needed, they must be declared after all the variables in the INCLUDE modules listed here.

"crtord, segord, remoteord, category, linkord" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual. These one-word integer values are some of the most crucial values in the TRAMCON software. They are all used as indices into arrays of data stored in the HEAP, discussed in Section 11.2, and their respective ranges are determined by constant definitions that are specified in the INCLUDE module [RECR3.

Any program that does terminal I/O determines which terminal it will communicate with by the value of "**crtord**" that is passed from program to program as the fourth run-string parameter "**parms[4]**". The value of "**crtord**" ranges from 0 to 4 ("**max_crts_per_master**" - 1) and is used as an index into the two HEAP arrays "**heap^.master^.crt_ptr[crtord]^**" and "**heap^.current_crt[crtord]**".

The value of "**segord**" is passed to display programs by program CMMD as the fifth run-string parameter, "**parms[5]**", to indicate for which segment the data is to be displayed. The data collection programs attach the value of segord to the remote unit CLASS I/O. The value of "**segord**" ranges from 0 to 1 and is an index into the HEAP arrays "**heap^.master^.segment[segord]**" and "**heap^.segment_status[segord]**".

The variable "**remoteord**" goes hand in hand with "**segord**" and references data one level deeper to the remote unit level. The value of "**remoteord**" ranges from 0 to 20 ("**max_remotes_per_segment**" - 1) and is an index into the HEAP arrays "**heap^.master^.segment[segord].seg_ptr^.remote_info[remoteord]**" and "**heap^.segment_status[segord].remote_status[remoteord]^**".

The value of "**category**" is usually computed by the particular program as it executes. The variable "**category**" goes hand in hand with "**segord**" and "**remoteord**" and references data one level deeper to a particular portion of a remote unit. The value of "**category**" ranges from -1 to 3 ("**max_linkends_per_remote**" - 1) and is an index into the HEAP arrays "**heap^.master^.segment[segord].seg_ptr^.
remote_info[remoteord]^.linkend_info[category]^**" and "**heap^.segment_status[segord].remote_status[remoteord]^.
cat_status[category]**". The value "**linkord**" has the same meaning as "**category**" and may not be widely used.

"sname_len" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual. This one-word integer value indicates the length of a given site name, in characters, and is used by programs that display the site names. This value is returned as the value of the function "read_dict" when it is called.

"local_end" - This one-word integer value is used by programs that access the link-end-specific information in the arrays "specific_name" and "relay_exists" in the LINKEND record. This integer assumes the two values 0 or 1 and is used as the first dimension index into the two-dimensional arrays just mentioned. The routines "get_category", located in the INCLUDE module [EXTNT, and "set_cat_vars", located in library $TRLIB, set this value along with other variables that are used to access data for a given linkend category.

"rnrq_status" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value indicates the result of a call to one of the system resource number routines. Resource numbers are used to control use of the terminal devices. The value of "rnrq_status" is returned as the value of the function when the routine RNRQ is called. Refer to the RTE-6/VM Programer's Reference Manual, pp. 5-16, for possible values for "rnrq_status" and their meanings.

"crt_type" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value ranges from 0 to 4 and represents the type of terminal device with which a program is communicating. These terminal types have been given alphanumeric identifiers in the CONST section of the INCLUDE module [RECR3. Each terminal type has certain features that are different or nonexistent on other types. Examples of the features are graphics grid dimensions, color, and alphanumeric memory size. Display programs retrieve the terminal type from the Configuration data in the HEAP and place it into "crt_type" because "crt_type" is easier to access. That is, the value "crt_type" is in the program space and is, therefore, a one-word, one-step de-reference while the value "heap^.master^.crt_info[crtord]^.terminal_type" is a three-step de-reference with each step evaluating a two-word HEAP address.

"trunkord" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value ranges from 0 to 99 ("max_trunks_per_segment" - 1) and is used as an index into the HEAP array "heap^.master^.segment[segord]^.trunk_info[trunkord]^". This value is used by the programs PC and PH to display the PCM or digroup alarms, by the programs MTRP and PLRP to process the remote unit responses that contain the PCM alarms and by the program HR to store each hour's PCM alarms in the disc file (PHIST.

"last_ln, first_ln, ln" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. These one-word integer values are used by various display programs to keep track of how much and where data are displayed on the terminal display. They are also used as a dereferencing convenience like that described above for "crt_type".

"time_alfa" - The type, "time_str", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This 12-character string value is used primarily by program CMMD to hold the ASCII representation of the date/time in the format displayed in the upper left-hand corner of all TRAMCON displays.

"print_it, refresh, colored" - The type, "BOOLEAN", is a Pascal basic type. These BOOLEAN values are used to control actions of the display programs. The display program sets "print_it" according to the value of the HEAP variable "heap^.current_crt[crtord].remotes_to_print", which was set by program CMMD just before it scheduled the display program. If "print_it" is true, the output is routed to the printer instead of to the terminal display. The value of "refresh" determines whether the display program displays the static information or just the dynamic data for a given display. The value "colored" is another convenience variable set to the same value as the HEAP variable "heap^.current_crt[crtord].color_crt" and determines whether the display program will or will not issue escape sequences to activate the color feature of the display device. Any display program can count on these values being correct because they are set by the routine "allocate_EMA", which is called by each program as one of the first execution steps.

"oppo_site, siteptr" - The type, "site_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. These two-word integer values are more dereferencing convenience values that are set by routine "allocate_EMA" and are HEAP addresses that point to site records. The primary site record of interest to a display program, such as the location of a given remote unit, is pointed to by the value "siteptr". The site record of the site at the other end of a particular link is stored in "oppo_site". Refer to Section 8.2.4.1 for a discussion of routine "allocate_EMA".

"equip" - The type, "equipment_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a Configuration data base equipment record and is another dereferencing convenience.

"crtptr" - The type, "crt_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a Configuration data base CRT record and is another dereferencing convenience.

"rem_status_ptr" - The type, "remote_status_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a dynamic remote status record and is another dereferencing convenience.

"remptr" - The type, "remote_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a Configuration data base remote record and is another dereferencing convenience.

"segptr" - The type, "segment_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a Configuration data base segment record and is another dereferencing convenience.

"cat_status_ptr" - The type, "current_alarm_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a dynamic category status record and is another dereferencing convenience.

"linksptr" - This one-word integer value is used as an index into the LINKS array. Every LINKEND record has one of these pointers to associate the particular LINKEND with the information contained in the LINKS array that is common to both ends of the given link. This variable is set by the routine "get_category" in [EXTNT and by routine "set_cat_vars" in library $TRLIB.

"linkendptr" - The type, "linkend_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a Configuration data base link-end record and is another dereferencing convenience.

"trunkptr" - The type, "trunk_record_ptr", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This two-word integer value is a HEAP address of a Configuration data base trunk record and is another dereferencing convenience.

"crt_buff" - The type, "cmd_str", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This 80-character string value is used by the routine "**keypress**" ($TRLIB) to store input from any keyboard. This input is attached to the CLASS number for the given keyboard (heap^.current_crt[crtord].kybrd_class) and can be recovered by any program that references that CLASS number.

"crt_IO_len" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value is used by programs to determine the length in characters of input from any given keyboard. The routine "**keypress**" uses this variable in conjuction with "**crt_buff**" described above. The actual input is placed into "**crt_buff**" and the length of the input (in bytes) is placed into "**crt_IO_len**".

"outunit" - The type, "TEXT", is an HP/1000 Pascal enhanced basic type. The text file is the only output file used by the TRAMCON programs to output to the display or the printer. To speed up the output to the display, the file size is adjusted to approximately one screen full by the compiler directive $LINESIZE 1920$. The directive $LINESIZE 128$ appears immediately following the declaration of "outunit" so that any other text files declared in any program will have the default 128-character buffer allocated. By increasing the buffer size, many of the terminal output statements can be LOGICAL (buffered) instead of PHYSICAL, thus reducing the number of time-consuming terminal setup steps required.

Figure 57 shows the INCLUDE file [DTVAR, which looks remarkably like file [TRVAR, which was discussed above. They look alike because they are the exact same definition except for the $LINESIZE$ directive in front of the declaration of the text file "**outunit**". The programs DT and SR are very large because they use the space-consuming DS routines. If a large buffer for the "**outunit**" file is added, the number of segments for DT or SR becomes undesirable. Therefore, a separate global VAR definition is maintained for use by DT and SR only.

```
VAR heap: heap_ptr; parms: parm_array; id: byte;
    word, sname: dictionary_word;  soft_key, color: CHAR;
    i, j, k, crtord, segord, remoteord, category, linkord, sname_len,
      local_end,rnrq_status,crt_type,trunkord,last_ln,first_ln,ln: INT;
    time_alfa: time_str; print_it, refresh, colored: BOOLEAN;
    oppo_site, siteptr: site_record_ptr; equip: equipment_record_ptr;
    crtptr: crt_record_ptr;    rem_status_ptr: remote_status_ptr;
    remptr: remote_record_ptr; segptr: segment_record_ptr;
    cat_status_ptr: current_alarm_ptr; linksptr: link_def_ptr;
    linkendptr: linkend_record_ptr; trunkptr: trunk_record_ptr;
    crt_buff: cmd_str;  crt_IO_len: INT;
    $LINESIZE 500$ outunit: TEXT; $LINESIZE 128$
```

Figure 57. Global data definition module [DTVAR.

The programs DT and SR include the file [DTVAR instead of file [TRVAR so that their buffer size for the file "**outunit**" is only 500 characters rather than 1920 characters. This line-size value in file [DTVAR can be adjusted, as either of these programs is changed, to cause the segmenter to create a reasonable number of segments.

CAUTION: If changes are made to the definition [TRVAR, the same changes must be made to file [DTVAR or programs DT and SR will not work properly. The first symptom will probably be the Pascal fatal error "**must open file ?**" that results from trying to access file "**outunit**" by the routine "**allocate_EMA**".

The global data definition shown in Figure 58 is separate from [TRVAR because, unlike the variables in [TRVAR, these variables are not used by most TRAMCON programs. In fact, these variables are used only by the programs MTRP, PLRP, CC, PA, and SW; by most of the routines in $MPLIB; and by a few routines in $TRLIB. Any module that references (INCLUDES) the VAR definition [MPVAR must insert it immediately after the inclusion of [TRVAR in order for the compiler to generate the proper addresses.

```
{ Ref by PLRP and MTRP, CC, PA, SW, $MPLIB, $TRLIB }
 VAR cos, polledord, node_idx, response_length, ssy, yr,
       sav_ord, response_status: INT;
     nodes: ARRAY[expression_ordinal] OF INT;
     pl_processor, res_len_ok, parityerr, bad_id, illegal_interrupt,
       response_timedout, responded, change_of_state, sw_response,any_new,
       any_just_cleared, new_gone, cleared_gone, new_cn, new_pc: BOOLEAN;
     cmd_buffer: six_chars; clk: parm_array;
     archive_file: FILE OF archive_record;
     unpacked_response: unpacked_response_record;
     response: response_str;
     major_or_minor: BOOLEAN; {TRUE if at least 1 major or minor alarm on}
     site_stat, ss_site_stat: two_chars; archive_rec: archive_record;
     alarms_reported: ARRAY[category_ordinal] OF BOOLEAN;
     init_2states: alarms_array;
     init_a2ds: a2ds_array; init_digitals: digitals_array;
{The following are used to process analog and digital parameter data }
     bin, decimal_places, two_sided_th, parmord: INT;
     decreasing, two_sided, in_a2ds, calibrate, crossed_amber,
       crossed_red, red_on: BOOLEAN;
     received_id: byte;
```

Figure 58. Global data definition module [MPVAR.

"cos" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value is used by the response processing routines in $MPLIB to represent the change-of-state indication in a remote unit response. The value of "**cos**" is set by routine "**get_answer**" in $MPLIB and is non-zero only if bit 5 of the second byte in the response is ON. This happens to be the way that the currently supported DATALOK10 remote unit indicates a change-of-state and, of course, may not indicate the same thing for another remote unit type that might be supported in the future. If a new remote unit were to be supported, this change-of-state processing would have to be generalized.

"polledord" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value is used by programs PLRP and MTRP to represent the same values as "remoteord" described above.

"node_idx" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value is used by routines "evaluate_node" and "process_response" in $MPLIB. The value is set by routine "evaluate_node" and is used by routine "process_response".

"response_length" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value is used by the routine "get_answer" (see Section 8.2.4.2) to store the length (in bytes) of each remote unit response received. This value is compared with hard-coded valid lengths to set the GLOBAL VAR "res_len_ok".

"ssy, yr, sav_ord" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. The variable "ssy" is no longer used and can be removed. The variable "yr" is referenced by routine "TimeNow" in $TRLIB and by routines "archive_it" and "get_answer" in $MPLIB. The value set by routine "get_answer" is used by routine "archive_it" for the archive record time stamp. Variables "ssy" and "sav_ord" do NOT appear to be referenced.

"response_status" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This one-word integer value is used by the remote unit response processing routines in $MPLIB and the main portions of programs PLRP and MTRP to reflect the most severe status of each response processed.

"nodes" - The type, "INT", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This array of one-word integer values is used by the recursive routine "evaluate_node" to hold the nodes of a combination alarm expression as it is evaluated. Routine "evaluate_node" is called by routine "process_response", which is in turn called by the programs MTRP and PLRP.

"pl_processor" - The type, "BOOLEAN", is a Pascal basic type. This one-word logical value is used by programs to control the behavior of the routines in the library $MPLIB. The flag "pl_prpcessor" is set to true by the program PLRP and set to false by any other program. This flag is checked by the routines "pm_Initialize" and "update_cursor" in $MPLIB.

**"res_len_ok, parityerr, bad_id, illegal_interrupt,
response_timedout, responded, change_of_state, sw_response, any_new,
any_just_cleared, new_gone, cleared_gone, new_cn, new_pc"**

The type, "**BOOLEAN**", is a Pascal basic type. These one-word logical values are used by the response processing code to define the characteristics of each response received from a remote unit. These variables determine the overall site status that is displayed to the left of each station on the segment MAP and segment status displays. The variable "**res_len_ok**" is set by the routine "**get_answer**" in $MPLIB and indicates that the response length is within the general range of 2 to 270 ("**max_chars_per_response**" defined in [RECR3). It is referenced three more times in the same routine. One of those references is to set the flag "**responded**" and another is to set the flag "**bad_response**". The only other place that "**res_len_ok**" is referenced is in routine "**process_response**" in $MPLIB.

The three values, "**parityerr**", "**illegal_interrupt**", and "**response_timedout**" are reported directly by the interface driver as individual bits in the value of "**response_status**" returned by the call to routine "**get_IO_length**" (alias "**ABREG**", refer to RTE-6/VM Programer's Reference Manual, p. 2-12). The value "**parityerr**" is a hardware flag from the BACI interface indicating that at least one byte in the response had incorrect PARITY. The "**illegal_interrupt**" flag is set by the driver when no valid reason for being in the driver could be found. The "**response_timedout**" flag is set by the driver when too much time elapses while waiting for a byte to show up at the BACI input buffer.

The "**bad_id**" flag is set by the routine "**get_answer**" and referenced by the routine "**get_answer**" and by the program MTRP. If the program is PLRP, "**bad_id**" is true if "**received_id**" does not equal "**remptr^.remote_polling_id**", that is the ID of the remote unit just polled. In the monitoring case, the program MTRP does not know which remote unit was polled and, therefore, can not match against a known ID. For program MTRP, "**bad_id**" is true if "**response_timedout**" or a search through all the remote unit IDs on the given segment for the "**received_id**" does not produce a match.

The variable "**responded**" is set by routine "**get_answer**" by LOGICALLY ANDing several of the other flags. This flag is also set by routine "**unpack_response**" if a DELETE character appears anywhere in the response other than the last character. The variables "**change_of_state, any_new, any_just_cleared new_gone, cleared_gone**" are referenced solely by the routine "**process_response**".

The variables "**new_cn**" and "**new_pc**" are set by routine "**process_response**" when at least one counted two-state and at least one PCM digroup alarm are detected respectively. The routine "**update_displays**" will not attempt to refresh any CN or PC display if the corresponding flag, either "**new_cn**" or "**new_pc**", is false. There appear to be no references to the variable "**cmd_buffer**".

"clk" - The type, "parm_array", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This array of five one-word integer values is used to hold the time/date clock in the format returned by the routine "read_clock" (alias EXEC, function 11, refer to RTE-6/VM Programer's Reference Manual, p. 2-72). This time/date is set by routine "get_answer" the moment a response is received from a remote unit and represents the time/date stamp associated with that response. This time/date is used by routine "archive_it" to time-stamp archive records and by routine "process_response" to time-stamp changes-of-state of individual elements of the response.

"archive_file" - The type, "archive_record", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This file of "archive_record" is declared globally so that it may be initially opened by the routine "pm_Initialize" and subsequently used by the routine "archive_it". The function of the open statement for this file in routine "archive_it" is to cause the last record logically written to be physically written to the file. A more appropriate routine is POST, which is discussed in the RTE-6/VM Programer's Reference Manual, p. 3-82.

"unpacked_response" - The type, "unpacked_response_record", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This record variable is used by the response processing routines in $MPLIB to hold a generic unpacked form of each remote unit response. Routine "pm_Initialize" initializes "unpacked_response" when the programs MTRP and PLRP are scheduled. After that, the content of "unpacked_response" is overwritten by routine "unpack_response" each time a response is received from any remote unit (refer to Section 8.2.4.2 of this manual for discussion of "unpack_response"). Figure 59 shows the format of the GENERIC response. Refer to Appendix E for a detailed tabular description of the GENERIC response formats for the DATALOK10 models 1D and 1E remote units. The GENERIC response has three major divisions; (1) two-state information, (2) Analog-to-Digital parameter information, and (3) Digital or Pulse Count parameter information.

Each of these three major divisions contains information for each of 4 possible CATEGORIES (CATEGORY is defined in the DICTIONARY in this manual). This GENERIC response defines the limits imposed by the TRAMCON software on the information that can be reported by a single PHYSICAL remote unit. That is, a single remote unit can report information for 1 Site equipment category and from 1 to 3 link end categories. Currently, the DATALOK10 remote units report information for all possible categories.

The two-state alarm/status section accommodates 144 indicators per CATEGORY. Of this 144 total, only 72 can actually be reported by the remote unit. The remaining 72 two-state indicators are derived from other information reported by the remote unit. The 72 derived indicators are allocated and interpreted as follows

12 A/D threshold crossings (6 Amber, 6 Red)
44 Digital parameter threshold crossings (22 Amber, 22 Red)
16 Combination two-state alarms

Routines "**evaluate_node**", "**p61or62**", and "**process_response**" process the response in this generic form.

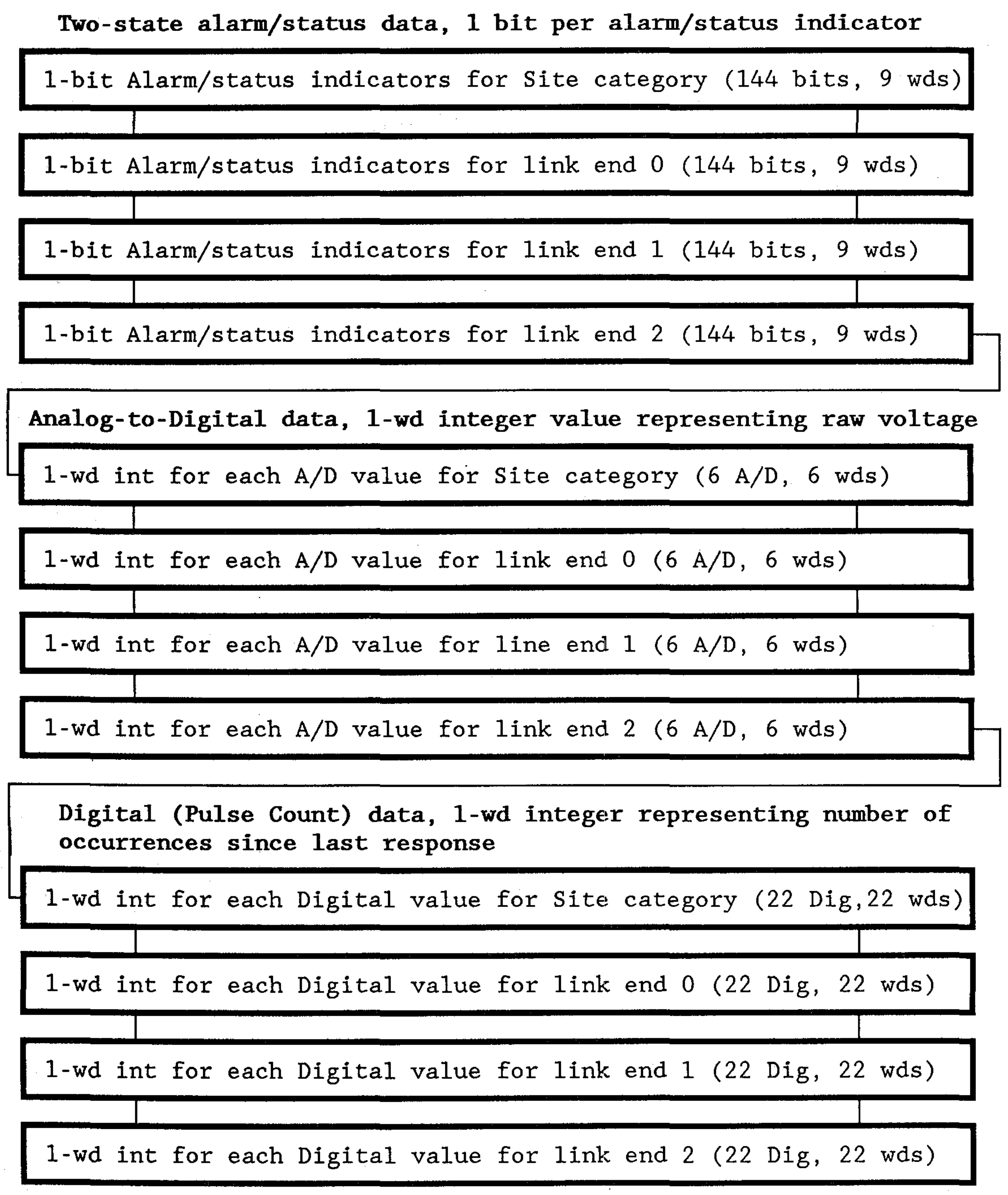

Two-state alarm/status data, 1 bit per alarm/status indicator
1-bit Alarm/status indicators for Site category (144 bits, 9 wds)
1-bit Alarm/status indicators for link end 0 (144 bits, 9 wds)
1-bit Alarm/status indicators for link end 1 (144 bits, 9 wds)
1-bit Alarm/status indicators for link end 2 (144 bits, 9 wds)
Analog-to-Digital data, 1-wd integer value representing raw voltage
1-wd int for each A/D value for Site category (6 A/D, 6 wds)
1-wd int for each A/D value for link end 0 (6 A/D, 6 wds)
1-wd int for each A/D value for line end 1 (6 A/D, 6 wds)
1-wd int for each A/D value for link end 2 (6 A/D, 6 wds)
Digital (Pulse Count) data, 1-wd integer representing number of occurrences since last response
1-wd int for each Digital value for Site category (22 Dig,22 wds)
1-wd int for each Digital value for link end 0 (22 Dig, 22 wds)
1-wd int for each Digital value for link end 1 (22 Dig, 22 wds)
1-wd int for each Digital value for link end 2 (22 Dig, 22 wds)

Figure 59. GENERIC remote unit response format.

"response" - The type, "response_str", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual. This 270 ("max_chars_per_response" defined in [RECR3) character string is used by the remote unit response processing routines in $MPLIB to hold the raw response from each remote unit. Routine "get_answer" places the response received over any segment polling channel into "response" by specifying "response" as the input buffer in a call to the routine "get_response" (alias EXEC, function 21, refer to RTE-6/VM Programer's Reference Manual, p. 2-41), which recovers the remote unit response that was attached to the CLASS number "caller" (either "plrp_class" or "mtrp_class"). Routine "print_response" in $MPLIB displays a formatted raw response directly from the variable "response". Routine "unpack_response" transforms the remote unit specific response in "response" into the generic "unpacked_response" GLOBAL variable.

NOTE

"major_or_minor" - The type, "BOOLEAN", is a Pascal basic type. This one-word logical value is used solely by the routine "update_displays" in $MPLIB to indicate that at least one MAJOR or minor alarm is present in the response just processed. It is set to TRUE if the first letter of the string "site_stat" equals "M" or "m", which means that the most severe status for the site whose response was just analyzed is either "MS" - Major Site, "ME" - Major Equipment, "mS" - minor Site or "mE" - minor Equipment.

NOTE

"site_stat, ss_site_stat" - The type, "two_chars", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1 of this manual These two 2-character strings are used solely by the routine "update_displays" in $MPLIB to hold the most severe site status indication for the site whose response has just been processed. The string "site_stat" is used to hold the site status indicator for the segment map display, while the string "ss_site_stat" is used to hold the site status indicator for the segment status (SS) display. The possible values of these strings are listed in Section 8.2.4.1 of this manual. The value of these strings is returned as the value of the VAR parameter when "update_displays" calls routine "get_site_status" (in $TRLIB).

"archive_rec" - The type, "archive_record", is defined in the INCLUDE module [RECR3 and is discussed in Section 11.1. This record value is used by routine "archive_it" in $MPLIB.

"alarms_reported" - This BOOLEAN array contains one BOOLEAN value for each possible category in a remote unit response. If any array element is true, at least one alarm/status indicator was reported for the corresponding category in the remote unit response just received. For the dumb DATALOK10 remote unit, these indicators are always true since all data for all categories are reported every time. This feature was intended to support a remote unit, such as the designed but not implemented IRU, that could selectively report information. The values are initialized to false by the routine "unpack_response" just before unpacking each response. The values are set to true by routine "unpack_response" if the remote unit type is a DATALOK10 model 1D or 1E.

"init_2states, init_a2ds, init_digitals" - The types, "alarms_array, a2ds_array, digitals_array", are defined in the INCLUDE module [RECR3 and are discussed in Section 11.1. These records are set to initial values once at TRAMCON startup by the routine "pm_Initialize" and are used by routine "unpack_response" to initialize the generic unpacked response record "unpacked_response" before unpacking any remote unit response. Initializing "unpacked_response" ensures that no residual data is left from the previous response.

11.4 Disc Files

This section describes the disc files used by TRAMCON for data storage and support of the TRAMCON On-Line system. The TRAMCON field system uses the HP-7912 65 Mbyte disc system, which is divided into two main sections called **"cartridges"**. Each cartridge is associated through the system generation with a logical unit (LU) number. The software communicates with the disc using this LU number. The LU numbers assigned to the two disc cartridges are 2 and 10. There is an alternate name, called the cartridge reference number (CRN), that can be used to reference the disc cartridges. The CRNs were chosen to be the same as the LU numbers to avoid any unnecessary confusion.

The main guideline used to locate files on the disc is:

PROGRAMS (INCLUDING OPERATING SYSTEM) ON LU 2
NONPROGRAM FILES ON LU 10.

Figure 60 lists the files that should be present on disc cartridge 2 (LU 2) of every TRAMCON field system.

All of the programs marked with an asterisk are not essential for TRAMCON operation, but are very useful for diagnosing problems that will continue to occur for the life of the system. These programs are also very useful for determining where adjustments should be made to increase the efficiency of the TRAMCON On-Line software system.

All programs marked with a **"D"** are distributed systems (DS) modules that support the InterProcessor Communication (IPC) function. This function supports a network with each TRAMCON master being a node on that network.

All programs marked with a **"U"** are utility programs that perform some cleanup, such as packing the disc cartridges (PAKLU) and closing files that were left open (CLNUP). They also set some flags in the date file (SETDT) and set some flags in the operating system disc cartridge table (SETCR). The program SETCL can be used to set the software clock from the hardware clock or vise versa.

	File Name	File Type	Disc Blocks	Record Length	Security Code
D	#SEND	6	8	128	0
	$SYENT	1	126	128	-22738
	(DATE	3	1	128	0
	)MISC	4	2	128	0
	+@CCT!	1	28	128	-31178
	AL	6	193	128	0
	ARPTR	6	98	128	0
	BROADC	6	82	128	0
	CC	6	163	128	0
	CF	6	438	128	0
	CHECKT	6	14	128	0
U	CLNUP	6	122	128	0
*	CMD	6	102	128	0
	CMMD	6	235	128	0
	CN	6	141	128	0
	CO	6	141	128	0
	CR	6	156	128	0
*	DCODE	6	60	128	0
D	DINIT	6	69	128	0
D	DLIST	6	29	128	0
*	DLX	6	218	128	21583=TO
D	DSINF	6	89	128	0
	DT	6	889	128	0
	ED	6	154	128	0
D	EDITR	6	81	128	0
	EDITT	6	305	128	0
D	EXECM	6	29	128	0
D	EXECW	6	61	128	0
	FC	6	52	128	0
	FC000	6	89	128	0
	FC001	6	111	128	0
	FC002	6	149	128	0
	FC003	6	140	128	0
	FC004	6	32	128	0
	FC005	6	87	128	0
	FC006	6	59	128	0
*	GENIX	6	140	128	0
*	HELP	6	44	128	0
	HI	6	139	128	0
	HR	6	199	128	0
	INIT	6	924	128	0
	KYBRD	6	83	128	0
	LO	6	154	128	0
	LOF	6	51	128	0
	LON	6	78	128	0

Figure 60. Contents of disc LU 2 for TRAMCON field system.

	LS	6	177	128	0
	LST	6	72	128	0
*	LUPRN	6	90	128	0
*	LUQUE	6	9	128	0
	MA	6	119	128	0
	ME	6	155	128	0
U	MEDX1	6	98	128	0
U	MEIDX	6	101	128	0
	MS	6	101	128	0
	MSG	6	104	128	0
	MTRP	6	385	128	0
	NP	6	106	128	0
D	OPERM	6	19	128	0
	PA	6	143	128	0
U	PAKLU	6	86	128	0
	PC	6	112	128	0
	PF	6	126	128	0
	PH	6	195	128	0
	PLRP	6	385	128	0
	PM	6	127	128	0
	POLL	6	50	128	0
	PR	6	100	128	0
D	PROGL	6	57	128	0
D	PTOPM	6	58	128	0
D	REMAT	6	86	128	0
D	RFAM	6	71	128	0
	RMASTE	6	67	128	0
D	RSM	6	23	128	0
	SC	6	150	128	0
D	SCOM	6	236	128	0
	SE	6	118	128	0
U	SETCL	6	69	128	0
U	SETCR	6	46	128	0
U	SETDT	6	61	128	0
	SI	6	101	128	0
	SR	6	482	128	0
	SS	6	121	128	0
	SW	6	185	128	0
D	SYSAT	6	31	128	0
	TIMPAS	6	94	128	0
	TIMSET	6	62	128	0
	TROFF	6	109	128	0

Figure 60. (cont.)

TS	6	114	128	0
UP	6	49	128	0
US	6	130	128	0
D VCPMN	6	8	128	0
WELCOM	3	3	128	0
WZ	6	108	128	0
X	6	59	128	0

D - DS modules
U - Utilities
* - TRAMCON diagnostics

Figure 60. (cont.)

All but a few of the files on LU 2, as shown in Figure 60, are type 6 program files. This is in keeping with the general file placement rule stated above, which says to place programs on LU 2 and data files on LU 10. The only files on LU 2 that are NOT type 6 are

$SYENT	1	126	128	-22738
(DATE	3	1	128	0
)MISC	4	2	128	0
+@CCT!	1	28	128	-31178
WELCOM	3	3	128	0

Each of these five non-type 6 files must be located on the system disc LU 2. Three of the files, $SYENT, +@CCT!, and WELCOM, are used by the RTE operating system which looks for them on the system disc cartridge (LU 2). The other two files, (DATE and)MISC, were created by the TRAMCON software developers and need to be accessed even when the TRAMCON software is down.

The File Copy utility (FC) MUST be present because, currently, it is the only way to distribute updates to the Configuration data base.

Figure 61 lists the files that should be present on disc cartridge 10 (LU 10) of every TRAMCON field system.

The data files marked with a "C" in Figure 61 belong to the current set of Configuration data base files. This is the data base that is used by the On-Line software. The files marked with an "O" belong to the old or fallback Configuration data base, which can become the current set by selecting the fallback option using the CO command. The files marked with an "N" are the new data base files, which can become the current set by selecting the new option using the CO command. It is this new set that is overwritten with each Configuration data base distribution tape. The files marked with an "R" are the run-time data base files.

	File Name	File Type	Disc Blocks	Record Length	Security Code
	"CM	4	111	variable	0
	"CM1	4	9	variable	0
	"CMIDX	2	18	2250	0
	"HE	4	106	variable	0
	"HE1	4	2	variable	0
	"HEIDX	2	18	2250	0
R	(ARCH	2	25216	125	21586=TR
R	(CN	2	117	356	21586=TR
C	(CRT	15	1	6	2810
R	(CURVE	2	273	832	21586=TR
R	(DATE	3	1	37	21586=TR
C	(DICT	15	55	7000	2810
C	(DINIT	15	2	variable	2810
C	(EQT	15	181	2313	2810
R	(HIST	2	16128	16	21586=TR
C	(LINK	15	23	119	2810
C	(LINKS	15	18	1750	2810
C	(MAST	15	1	47	2810
	(MISC	4	2	variable	0
C	(NET	15	2	140	2810
R	(PF	2	41	12	21586=TR
R	(PHIST	2	75	4800	21586=TR
C	(REMOT	15	2	15	2810
R	(RR	2	120	160	21586=TR
	(RUVER	4	11	variable	0
R	(SCIDX	2	246	630	21586=TR
C	(SEG	15	2	244	2810
C	(SITE	15	9	9	2810
R	(STATZ	3	9	650	21586=TR
C	(TRUNK	15	10	76	2810
N	)CRT	15	1	6	2810
N	)DICT	15	55	7000	2810
N	)DINIT	15	2	variable	2810
N	)EQT	15	145	2313	2810
N	)LINK	15	13	119	2810
N	)LINKS	15	18	1750	2810
N	)MAST	15	1	47	2810
N	)MISC	4	2	variable	0
N	)NET	15	2	140	2810
N	)REMOT	15	1	15	2810
N	)SEG	15	2	244	2810
N	)SITE	15	9	9	2810
N	)TRUNK	15	15	76	2810
O	^CRT	15	1	6	2810
O	^DICT	15	55	7000	2810
O	^DINIT	15	2	variable	2810

Figure 61. Contents of disc LU 10 for TRAMCON field system.

O	^EQT	15	181	2313	2810
O	^LINK	15	41	119	2810
O	^LINKS	15	18	1750	2810
O	^MAST	15	1	47	2810
O	^NET	15	2	140	2810
O	^REMOT	15	3	15	2810
O	^SEG	15	4	244	2810
O	^SITE	15	9	9	2810
O	^TRUNK	15	36	76	2810
	COOFF	4	1	variable	0
	STOFF	4	3	variable	0

Figure 61. (cont.)

The data in these files are updated by the TRAMCON On-Line software. Note that all Configuration data base files are type 15 and have security code set to 2810. All of the run-time data base files have a security code of "TR". This allows the user to manipulate these files in groups by using the common types or security codes.

11.4.1 Changing the Record Size

This section is extremely important because it raises the caution flag to the software developer when changes are considered that would affect the size of the records in either the Configuration data base files or the run-time data base files. Seemingly insignificant changes become difficult to implement if they cause a change in the record size of any of these files. Most of the I/O to these files in the TRAMCON On-Line software is done with standard Pascal I/O statements. Pascal I/O will NOT read a record from any of these files that is a different length (larger OR smaller) than the record that was written to the file. The telltale symptom is the **FATAL** run-time error message:

```
PASCAL I/O ERROR ON FILE xxxxxx
SEQUENTIAL ACCESS READ ERROR
```

where "xxxxxx" is the Pascal logical file name. This message says that the logical file definition in the Pascal program where the FATAL error occurred does NOT match the physical file definition.

Changes to the CONSTANTS and TYPE definitions should not be avoided because they might affect a disc file record size. On the other hand, one must exercise CAUTION when making changes that affect the records in these files. In Section 11.1, in discussing the TRAMCON CONSTANTS and TYPE definitions, there is an attempt to indicate whether changing a particular CONSTANT or TYPE would affect any record sizes for any of these disc files. If so, the reader is cross-referenced to this section for general guidelines on how to implement such a change to a disc file record.

Basically, the method for implementing record changes for Configuration data base files is the same as that for the Run-time data base files. Pascal allows records of a different size than is currently in the file to be written to disc files. The solution is to create a program that can write the new record to a file before any program attempts to read the new record. That is easier said than done because, in most cases, one wants to preserve the data in the old records. In order to preserve the already existing data, the file must be opened and associated with the old record format, each record must be read with the old format, the existing information must be transferred to a new format record and that new record must be written to a file that is associated with the new record format. One small consolation, in a file that has only one record, the two files involved can be the same file. Otherwise, a new file must be created to accept the new records and, at completion, the old file must be purged and the new file renamed.

Appendix D contains a listing of a program that can be used to alter the records in a disc file as described in the preceeding paragraph. The example in Appendix D was used to change the size of the records on the disc file (STATZ. Only minor modifications need to be made to the listing in Appendix D to make the program work for a different disc file. There are seven points in the source listing where code must be changed to accommodate a different disc file. These file-specific lines of code are bracketed in front by a COMMENT that begins with the word ENTER and followed by the COMMENT **{end ENTER}**.

The first file-specific piece of information is the new record length in machine words. This value, referred to as **"newreclen"** in the code, can be determined for simple records by counting the words in the definition. The size of more complicated records can be determined by removing the COMMENT brackets from the **{$LIST ON, TABLES ON$}** line near the end of the listing and compiling the program with the statement: **RU,P,&CHREC,-,%CHREC::10**. The new record definition must have already been entered into the source code. To find the new record length, simply scan through the listing file **'CHREC** to find the compiler table entry for the record variable type **"new_record"**. The entry should look similar to:

```
          NEW_RECORD
                                <125> Type
```

This information refers the reader to the TYPE definition with the internal identifier **<125>**. Look for a line similar to:

```
          <125>                 670/ 0 Record
```

This line indicates the size of record type **<125>** in whole machine words plus any excess bits less than one complete word. In this example, record type **<125>** is 670 words + 0 bits long. The value 670 should be placed into the CHGREC source code as the value of CONST **"newreclen"**.

Program CHGREC, listed in Appendix D, uses File Manager I/O routines rather than Pascal routines because File Manager offers greater control and flexibility in dealing with disc files on this system. This program declares a Data Control Block (DCB) for an input file (olddcb) and a DCB for an output file (newdcb). The old file is OPENED normally and the new file is CREATED with the temporary name XXXXXX on the same disc cartridge as the old file. If the file type is greater than 2 (variable record length), the remainder of the disc cartridge is reserved for the new file to prevent the creation of extents for the new file. If the file type is 1 or 2 (fixed record length), the total number of records must be known so that the actual file size can be calculated. The program reads each record from the old file, transfers the existing data from the old record (old_rec) to the new record (new_rec), possibly initializes some new data in the new record and writes the new record to the new file (XXXXXX). When all the records have been transferred (reached EOF on old file), the old file is CLOSED and PURGED and the new file CLOSED and RENAMED from XXXXXX to the old file name. If the file type is greater than 2 (variable record length), the file is TRUNCATED to its actual size.

The only difference between the Configuration files and the Run-time files is the stage at which the switchover takes place. That is, with the Configuration files, the switchover must be done prior to any Configuration with the new records and both the Configuration software and the On-Line software are affected. A change that only affects the records in the Run-time files will only affect the On-Line software. Of course, there are changes that affect both Configuration files and Run-time files.

At the same time, the On-Line software and/or the Configuration software is recompiled and reloaded with the new record definitions. Once the changeover program has been run, the TRAMCON software can use the files with the new record definitions.

11.4.2 Configuration Data Base

The Configuration data base is a collection of static data used by the On-Line TRAMCON software to describe the operational environment for any particular TRAMCON master computer. This data base is typically generated by some central site, remote from any particular TRAMCON master, and distributed to each master by magnetic tape. This data base is static in the sense that it is read at initial On-Line startup and is never altered by the On-Line software. Although, once the TRAMCON system becomes mature, it is not anticipated that many changes to this data base will be required, though a few changes will always be necessary. Therefore, there is a scheme for incorporating future updated versions of this data base.

The Configuration data base consists of 12 record types that are stored in 12 corresponding disc files. Essentially, three copies of these 12 disc files are kept on each TRAMCON master computer. One set of 12 files is currently used by the On-Line software. The other two sets are for the fallback and change-to-new data base functions. As with all disc files used in the TRAMCON system, there is a naming convention used for these Configuration data base files. The backup set's file names begin with character "^". The current set's names begin with character "(", and the new set's names begin with the character ")". The remainder of the name for each of the 12 files is the same for all 3 sets. The 3 sets of 12 Configuration data base disc files are listed in Figure 62. When an updated version of the data base is received, it is stored on disc replacing the 12 disc files whose names are spelled as listed in Figure 62 with the character ")" as the first character of each name. For example, the new disc file name for the dictionary file is ")" plus "DICT", resulting in the name ")DICT". For ease of handling these sets of data base files, they all have the same security code of 2810 and they are all stored on logical unit number 10 of the disc. This, along with the naming convention, allows the user to store and retrieve all files with the same security code, whose names begin with ")", "(", or "^" and that reside on LU 10.

11.4.3 Run-time Data Base

The Run-time data base consists of the 10 disc files listed in Figure 63. All files must be located on disc cartridge 10, must have security code 2810, and must be File Manager type 2 (fixed record length) files. Also notice that the naming convention chosen uses a "(" as the first letter in the file name.

Current	New	Backup
(DICT	)DICT	^DICT
(NET	)NET	^NET
(LINKS	)LINKS	^LINKS
(MAST	)MAST	^MAST
(LINK	)LINK	^LINK
(REMOT	)REMOT	^REMOT
(SEG	)SEG	^SEG
(TRUNK	)TRUNK	^TRUNK
(EQT	)EQT	^EQT
(CRT	)CRT	^CRT
(SITE	)SITE	^SITE
(DINIT	)DINIT	^DINIT

Figure 62. List of configuration data base disc files.

1.	(ARCH	- Archived Alarm/Status
2.	(CN	- Counted 2-state Information
3.	(CURVE	- Signal Quality Parameter Calibration Curves
4.	(DATE	- Miscellaneous Run-time Data
5.	(HIST	- Signal Quality Parameter Histograms (24 hour)
6.	(PF	- Power Fail Messages
7.	(PHIST	- PCM Histograms (Passed 24-hour Digroup Alarm History)
8.	(RR	- Simulator Canned remote unit Responses
9.	(SCIDX	- Scenario File Index
10.	(STATZ	- Run-time Statistics

Figure 63. Run-time Data Base Files, All on disc LU 10.

The Run-time data base consists of 10 separate disc files that are currently sized to hold the data for 42 remote units. The largest file is the parameter histogram file (HIST. Currently, the data base can accommodate two active segments of up to 21 remote units each. The data base files can be initialized using the program CF to set all files except (CURVE and (RR to zeros. Caution should be used when initializing (CURVE since this file may have been maintained in the field with great care. The data are stored with no wasted space for nonexistent remotes. That is, although there can be up to 21 remote units per segment, space is allocated for only those remote units that are defined. This method wastes no space but is very rigid. For instance, to add a remote to a segment is a relatively easy task for a trained person to do, using the Configurator. The tightly packed archive file is not so accommodating. To preserve all archive data, the data for all remote units following the point of insertion must be moved.

1. (ARCH

This file contains a fixed amount (currently 200) of past alarm/status records for each active remote unit. The referencing method is similar to that described above for file (ALARM. The 200-record portion of the file for each remote is treated as a circular FIFO file. Once 200 records have been collected for any given remote, the next record collected for that remote will overwrite the oldest record for that remote unit. The time spanned by the 200 records for a given remote unit depends on the alarm activity of that remote unit. For instance, if a chronic alarm causes an archive record to be created during each poll, then the oldest record could be only minutes or hours old. If a remote unit encounters only a few alarms each week, then the archived information for that remote unit could go back months. Also, chronic problems will affect the archived information for that remote unit only. The following FMGR command can be used to CReate the disc file (ARCH:

```
    CR,(ARCH:TR:10:2:8400:125
```

RECORD Description: length = 125 words, defined in [RECR3
initialized by CF

```
    archive_alarm_status_record = {119 wds}
      RECORD
      arch_year: INT;
      arch_jday: nine_bits;
      arch_hour: seven_bits;
      arch_minute, arch_second: byte;
      archive_alarms: {114 wds}
         PACKED ARRAY[category_ordinal{4},link_2state_ordinal{144}] OF
           PACKED RECORD {3 bits}
           arch_just_cleared, arch_new_alarm,arch_alarm_on: BOOLEAN
           END;
      END;
    archive_idx_record = ARRAY[1..124] OF INT; {124 wds}


    archive_record = {125 wds, record for disc file (ARCH}
      RECORD
    CASE ar_rec_type: BOOLEAN OF
     FALSE: (arch_idx: archive_idx_record); {124 wds}
      TRUE: (arch_rcd: archive_alarm_status_record) {119 wds}
   END;
```

The "**archive_record**" definition shown above can be found in INCLUDE module [**RECR3** and is the definition of the records for disc file **(ARCH**. Note that the definition actually consists of two alternate definitions, "**arch_idx**" and "**arch_rcd**". As described above, the archive file is treated as several subfiles, one for each remote unit defined in the data base. (Each of these subfiles is a circular FIFO file of "**max_archive_record**" many records). The first record in the archive file is of type "**archive_idx_record**". This record contains pointers or indices into these subfiles that indicate the NEXT available record number for each subfile. All other records in the archive file are of type "**archive_alarm_status_record**" and contain the actual archived information.

NOTE

Changes that will affect the record size for file (ARCH are

1. Changing the constant "**nbr_2states_per_link**"
2. Changing the constant "**max_linkends_per_remote**"

2. (**CN**

This file contains one record per remote unit defined in the data base. The record definition can be found in file [RECR3 and has the identifier "**cn_record**". A flag is stored for each two-state alarm monitored by the given remote unit. This flag indicates whether the number of occurrences of the given two-state value are being counted or not. If they are being counted, the start time/date (when the counting began) and the actual count (number of responses that showed the given two-state ON) are recorded in real time in the HEAP variable:

"**heap^.segment_status[segord].remote_status[remoteord]^.counts**".

The record "**counts**" holds the tally for all counted two-states for all categories for an entire remote unit. The two-states can be counted indefinitely, and therefore, it is necessary for the counts to be periodically saved to this disc file in case the TRAMCON master needs to be taken down and rebooted. The "**counts**" record for each remote unit defined in the data base is copied to this disc file every hour on the hour by the program HR. Therefore, outside of a disc head crash, the worst case would result in the loss of the last 59 minutes data. When the TRAMCON master is booted up, program INIT reads the records from this file and uses these data to initialize the HEAP "**counts**" records.

The following FMGR command can be used to CReate the disc file (CN:

CR,(CN:TR:10:2:117:356

```
RECORD Description:  length = 356 words, defined in [RECR3
                     initialized by CF
```

```
     counted_array = PACKED ARRAY[link_2state_ordinal{144}]OF BOOLEAN;{9 wds}
     counts_array = ARRAY[0..max_counts_per_link{20}-1] OF {80 wds}
       PACKED RECORD {4 wds}
       val, yr: INT;  jdy: nine_bits; hr: seven_bits; minut, secs: byte
       END;
     cn_record = ARRAY[category_ordinal{4}] OF {356 wds, record for file (CN}
       RECORD {89 wds}
       cn_vals:counts_array;{80 wds}
       cn_counted:counted_array {9 wds}
       END;
```

NOTE

Changes that will affect the record size for file (CN are

1. Changing the constant "nbr_counts_per_link"
2. Changing the constant "max_linkends_per_remote"
3. Changing the constant "max_2states_per_link"

3. (CURVE

This file contains the information to calibrate the analog parameters, reported in volts, to more useful units such as "dBm". Differences in equipment characteristics may also be accounted for by using these calibration curves. Sixteen ("nbr_bins") raw voltage values are kept for each parameter marking the 16 discrete calibration steps. Also stored in these records are the range (top and bottom) over which each parameter (analog and digital) can vary, and the amber and red threshold values for each parameter (analog and digital). Each 832-word record contains the calibration, range, and threshold values for an entire remote unit and is calculated as follows:

16 calibration words per analog parameter
6 ("max_a2d") analog parameters per category
4 categories (3 linkends + 1 site) per remote unit

PLUS

2 range and 2 threshold values per parameter
6 ("max_a2d") analog and 22 ("max_digital") digital parameters per category
4 categories (3 linkends + 1 site) per remote unit

The following FMGR command can be used to CReate the disc file (CURVE:

```
     CR,(CURVE:TR:10:2:273:832
```

RECORD Description: length = 832 words, defined in [RECR3
                     initialized by CF

```
     parm_record =  {832 wds, record for disc file (CURVE }
       RECORD       {category_ordinal=-1..2, a2d_ordinal=0..5 }
       cal_curves: ARRAY[category_ordinal{4}] OF
                    ARRAY[a2d_ordinal{6}] OF hist_array; {384 wds}
       a2d_bottom , a2d_top , a2d_amber , a2d_red:
         ARRAY[category_ordinal{4},a2d_ordinal{6}] OF INT; {96 wds}
       digital_bottom , digital_top , digital_amber , digital_red:
         ARRAY[category_ordinal{4},digital_ordinal{22}] OF INT {352 wds}
       END;
```

NOTE

Changes that will affect the record size for file (CURVE are

1. Changing the constant "nbr_bins"
2. Changing the constant "max_linkends_per_remote"
3. Changing the constant "max_a2ds_per_link"
4. Changing the constant "max_digitals_per_link"

4. (DATE

This file contains miscellaneous run-time information. It is small but absolutely essential for TRAMCON operation. Some data such as "**unused**", "**dmy1**", "**dmy2**", and "**dmy3**" have become unused during development, but are left in the "**date_record**" so that the record size for this file did not change, (see Section 11.4.1) and so that a few small items may be added as the need arises. The following FMGR command can be used to CReate the disc file (DATE: **CR,(DATE:TR:10:2:1:37**

RECORD Description: length = 37 words, defined in [RECR3, initialized by CF

```
     date_record = {37 words}
       RECORD
       yr, jdy, offset, heap_class_no, configuration_flag,unused: INT;{6 wds}
       version_date: INTEGER; {2 wds}
       version_nbr: REAL; {2 wds}
       segnames: ARRAY[master_segment_ordinal{4}] OF six_chars; {12 wds}
       nremotes: ARRAY[master_segment_ordinal{4}] OF INT; {4 wds}
       dmy1,dmy2,dmy3: INT; {3 wds}
       password: parm_array; {5 wds}
       time_serial_number: INT; {1 wd}
       message_serial_number: ARRAY[1..10] OF INT; {1 wd}
       logoff_class_no: INT; {1 wd, STOFF sets to 0 when ST command entered}
       END;                  {AUTOR checks if > 0 then TRAMCON is active}
```

NOTE

A change that will affect the record size for file (DATE is

Changing the constant "max_segments_per_master"

5. (HIST

This file contains a 24-hour history of the performance of all analog parameters (e.g., RSL) for every active link-end. Each record (of type "**hist_array**") is 16 words long and represents 1 hour's data for a single parameter. Each analog parameter is converted from a voltage to the proper units and binned as one of 16 discrete values. The 16 values are kept for each of the past 24 hours for each of 28 parameters (22 digital and 6 analog) for each category (3 link-end and 1 site) for each remote unit (up to 21) on every segment (up to 2) defined in the data base. The following FMGR command can be used to CReate the disc file (HIST:

```
CR,(HIST:TR:10:2:16128:16
```

RECORD Description: length = 16 words, defined in [RECR3
initialized by CF

```
hist_array        = ARRAY[1..nbr_bins{16}] OF INT;
```

Each 16-word ("**nbr_bins**") record in this file contains the 16 bin values for a given parameter for 1 hour. On the hour, program HR copies the past hour's information for each parameter defined in the data base from the HEAP variables

```
"heap^.segment_status[segord].remote_status[remoteord]^.
     cat_status[category]^.hist_a2d                          and

"heap^.segment_status[segord].remote_status[remoteord]^.
     cat_status[category]^.hist_digital
```

to the corresponding records in this file.

NOTE

A change that will affect the record size for file (HIST is

Changing the constant "nbr_bins"

6. (PF

This file contains time-stamped power fail messages. Each message is written to this file by program AUTOR. Any existing messages can be viewed by the TRAMCON operator using the PF command. Each 12-word record corresponds to one power fail event. The file currently holds 400 power fail messages. The following FMGR command can be used to CReate the disc file (PF:

```
     CR,(PF:TR:10:2:41:12
```

RECORD Description: length = 12 words, defined in [RECR3
initialized by CF

```
     pf_record = {12 wds, record for disc file (PF }
       RECORD
       on_year ,on_jday ,on_hour ,on_minute ,on_sec ,on_msec,
       off_year ,off_jday ,off_hour ,off_minute ,off_sec ,off_msec: INT
       END;
```

7. (PHIST

This file is a 24-hour history of the performance of the digroup alarms for all segments. No space conservation is necessary since this file is relatively small (only 75 disc blocks). Each 4800-word record contains the 24-hour history of all 100 possible trunks per segment for each of 2 trunk ends. The file is currently set to handle data for two segments or two 4800-word records. The following FMGR command can be used to CReate the disc file (PHIST: **CR,(PHIST:TR:10:2:75:4800**

RECORD Description: length = 4800 words, defined in [RECR3,initialized by CF

```
     pcm_histogram_array = {200 wds}
       ARRAY[1..2,0..max_trunks_per_segment{100}-1] OF INT;
     pcm_histogram_record = ARRAY[0..23] OF pcm_histogram_array; {4800 wds}
```

There is one large (4800-word) record of type "**pcm_histogram_record**" in file (PHIST for each segment defined in the data base. Each record contains 24 200-word arrays of type "**pcm_histogram_array**". Data are recorded by the program HR, on the hour, into the array element that corresponds to the current hour. Each of these arrays holds the digroup alarms for each end of each TRUNK (up to "**max_trunks_per_segment**") defined on a given segment for a given hour of the day. The digroup alarms are accumulated in the HEAP VAR "**heap^.segment_status[segord].pcm_counts**" for each segment defined in the data base.

NOTE

A change that will affect the record size for file (PHIST is

Changing the definition of "pcm_histogram_record" by
Changing the constant "max_trunks_per_segment"

8. (RR

This file contains simulated responses for each of the active remotes on a master. The program SI reads the remote response from this file if the remote response is being simulated. This file is currently sized for 33 remotes. Two responses are kept for each remote. One is the current response and the other is the default or start-over response. This default response is hard-coded in the program CF. The operator may set the current response equal to the default response anytime by using the proper SI command. The following FMGR command can be used to CReate the disc file (RR:

```
CR,(RR:TR:10:2:144:172
```

RECORD Description: length = 172 words, defined in [RECR3
initialized by CF

```
si_response_record =    {172 wds, Simulator response record, file (RR }
  RECORD
  request_error: INT; {1 wd, non-zero if remote unit detected error in
                             request received from TMT.
                              1. msg length limit exceeded.
                              2. command error.
                              3. category error.
                              4. number(s) out of range.
                              5. date-time error.
                              6. numbers NOT in ASCENDING order.
                              7. numbers duplicated.
                              8. count error.
                              9. action error.
                             10. unwired/unused error.
                             11. momentary control deactivation error.
                             12. configuration table error. }
  diag_error: INT;    {1 wd, non-zero if remote unit background
                             diagnostics discover an error.
                              1. main processor failure.
                              2. data acquisition failure.
                              3. memory board failure.
                           4-17. I/O card failure.
                         18-255. software fault. }
  diags: twenty_chars; {10 wds, one char per module in remote unit.
                               "o" - no CPU fault identified.
                               "m" - main CPU fault.
                               "a" - auxiliary CPU fault. }
  response_body: unpacked_response_record; {160 wds}
  END; {si_response_record}
```

NOTE

Changes that will affect the record size for file (RR are

1. Changing the definition of "unpacked_response_record" by
   changing the constant "max_2states_per_link"
   changing the constant "max_linkends_per_remote"
   changing the constant "max_a2ds_per_link"
   changing the constant "max_digitals_per_link"
2. Changing the definition to "twenty_chars"

9. (SCIDX

This file contains a list of all scenario files on this master. The sample scenario file SCEN00 is delivered with the TRAMCON system, and its name is entered in this index when initialized by the program CF. The following FMGR command can be used to CReate the disc file (SCIDX:

```
CR,(SCIDX:TR:10:2:5:630
```

RECORD Description: length = 630 words, defined in [RECR3
initialized by CF

```
sc_indexs_record = ARRAY[0..29] OF {630 wds, record for disc file (SC }
                    RECORD passwd:two_chars; {1 wd}
                    f_description:forty_chars {20 wds}
                    END;
```

10. (STATZ

This file has one record of type "**statz_record**", which contains system performance statistics. This record can be initialized to zeros by running program CF. The record definition "**statz_record**" is defined in INCLUDE module [RECR3. At boot-up, program INIT reads the contents of this record from disc file (STATZ and places these values into the miscellaneous shared memory record "**heap**" under name "**heap^.statz**". The information in the "**statz_record**" is updated in memory as it changes but it is permanently recorded on disc only once an hour, on the hour by the program HR. The information can be viewed on-line by entering command US. The data may also be initialized using the same command. The following FMGR command can be used to CReate the disc file (STATZ: **CR,(STATZ:TR:10:2:6:650)**

RECORD Description: length = 650 words, defined in [RECR3, initialized by CF

```
statz_record = {650 wds, record description for file (STATZ }
  RECORD
  cnt_cmds: ARRAY[un..us{46}, master_crt_ordinal{5}] OF INT;{230 wds}
  transmission:
    ARRAY[master_segment_ordinal{4}] OF
      ARRAY[segment_remote_ordinal{21},msg_status{5}] OF INT; {420 wds}
  END;
```

The single record in the STATZ file currently contains:

1. Operator command usage for each terminal keyboard.
   Name: heap^.statz.cnt_cmds[cmd,crtord]
   Updated by: CMMD in the HEAP as each command is entered.

2. TRAMCON remote unit transmission performance statistics.
   Name: heap^.statz.transmission[segord][remoteord,msg_status]
   Updated by: PLRP or MTRP as each remote unit response is received.
   "msg_status" has the following values:

   1. polls   - total poll msgs sent
   2. msg_ok - no transmit problem detected
   3. par_err - parity error detected
   4. bad_res - invalid response length or invalid remote id
   5. no_ans - no response received

NOTE

Changes that will affect the record size for file (STATZ are

1. Adding or deleting TRAMCON commands between "un" and "us"
2. Changing the constant "max_crts_per_master"
3. Changing the constant "max_segments_per_master"
4. Changing the constant "max_remotes_per_segment"
5. Changing the CLASS definition "msg_status"

12. RUN-TIME DIAGNOSTICS AND STATISTICS GATHERING UTILITIES

A diagnostic facility has been built into the software and can be invoked by entering the master password (see Section 12.1) to turn off the access restricted flag and then including the "**d**" option on any TRAMCON command. The "**d**" option has a toggle affect on the diagnostic flag just like the master password has on the access restricted flag. This means that each time the option is included in a command, the sense of the flag will change. That is, if the diagnostic flag is off and the "**d**" option is included in any TRAMCON command, the diagnostic flag will be set to on and vice versa.

Therefore, to turn diagnostics on and leave them on, just include the **"d"** option in any one command. The diagnostic flag will remain on for all successive operations until the **"d"** option is included in another TRAMCON command.

Remember, the access restricted flag must also be set to false in order for the diagnostic code to be activated. The access restricted flag and the diagnostic flag are kept in the shared data area called the HEAP. They are Pascal BOOLEAN values and are referenced by any program as **"heap^.access_restricted"** and **"heap^.diag"**, respectively. Sections 12.3 through 12.7 describe the use of this diagnostic feature by particular program modules.

The diagnostic code was some of the last to be added to the software and, in most cases, was added to diagnose a particular problem. Most of the diagnostic code is for the use of a professional software developer, not for the untrained. The explanations to follow are the only source for complete understanding of what some of the diagnostics are telling the user. Since this diagnostic feature is protected by the two flags mentioned above, there is minimal risk to the operational TRAMCON software when additions, corrections, or enhancements are made to this code.

12.1 Use of Passwords

There are three distinct uses of passwords in the TRAMCON On-Line software. To distinguish them for this discussion, they will be referred to as the LOGON, DT, and master passwords.

The LOGON password is currently in a non-implemented or hard-coded state. When an operator logs on to a remote terminal, a password is required. The intention was to require valid log-on IDs and valid passwords for security. The log-on program LO asks for a log-on ID and for a password. Currently, any input is accepted as a valid ID and the hard-coded password **"tr"** is accepted as the password. If tighter security is required in the future, this facility can easily be enhanced.

The DT password is used by all programs that transfer data from one master to another. Program DT requires this password when an operator chooses to send data to another master. The intention here is to protect any master's data from being improperly overwritten by someone who does not have permission and/or the knowledge to do so. Program SR requires this password when the operator wishes to broadcast the time/date to other masters, thus altering their system clocks. This password is stored in the disc file (DATE and can be changed by the operator using the PW command.

The PW command is processed by the program CMMD in routine **"process_simple_cmd"**. Also, the command parsing routine **"parse_it"** in CMMD allows the PW command to be entered from the system console only. When the operator enters the PW command with no further parameters, program CMMD displays the following prompt:

Password:

If the "**access_restricted**" flag is false, the value of the DT password will be displayed immediately after the above prompt. For example:

Password: TR

In the example above, the DT password consists of the two ASCII characters "**TR**". The operator must now enter the current password (in the example, the operator must enter "**TR**"). If the proper password is entered, the operator is prompted to enter a new value for the DT password as follows:

New Password:

The new value of the DT password will be the first 1 to 10 characters entered. The password is stored in the (DATE file in an encoded format so that it can not be read by looking at the record in the (DATE file.

NOTE

A small refinement should be made to this password with respect to the DT function of sending data TO another master. Rather than requiring that the DT operator know the local DT password, he/she should be required to know the DT password of the receiving master. This gives greater control to a TRAMCON master when data are being sent TO them.

The third and final password used by the TRAMCON software is the master password. This password is hard-coded in routine "**Initialize**" in program CMMD and its current value is: **e m" !eezz**

This password toggles the "**access_restricted**" flag described in Section 11.1.2. Each time the master password is correctly entered, the value of "**access_restricted**" is toggled from false to true or true to false. To enter the master password, the operator must first enter

pw,-1

Program CMMD responds with the prompt

Restricted Access is XXXXX, Enter Password to toggle:

where XXXXX is either "**TRUE**" or "**FALSE**".

As mentioned above, the value of the master password is hard-coded in routine "**Initialize**" in program CMMD, approximately line 872. To change the master password, the programmer simply changes the assignment statement in routine "**Initialize**" and recompiles and relinks the program CMMD. If more security is desired for this password, one simple step would be to encode/decode this password with the same routines used for the DT password.

The "**access_restricted**" flag controlled with this password is used to restrict use of certain TRAMCON commands. The commands that are allowed only if this flag is false are contained in the set of commands "**restricted_cmds**" in the program CMMD. The current value of "**restricted_cmds**" is set in routine "**Initialize**", approximately line 918, as follows:

```
restricted_cmds := [cf,cr,dn,eq,lo,lu,ms,off,ru,sc,sm,up,us];
```

The "**access_restricted**" flag is also used, in conjunction with the "**diag**" flag, to allow any software to display diagnostic information that can aid the software developer in diagnosing problems. The next two sections describe how some programs make use of this diagnostic feature.

12.2 Statistics Gathering (US command)

The program US currently gathers performance statistics to help the TRAMCON software developers and maintainers diagnose trouble spots, bottlenecks, and general software performance problems so that the software can be intelligently tailored to give maximum performance and operator responsiveness. Program US is protected by the "**restricted_access**" flag. It is NOT documented in the Operator's Manual because it is for diagnostic purposes only and does not perform a critical TRAMCON function.

Program US does not do the statistics gathering. Instead, it is used to initialize the statistics data, turn the statistics gathering code ON/OFF, and display the statistics that are gathered.

Currently, four sets of data are gathered and displayed by this US function. These are referred to as the COMMAND, TRANSMISSION, TIMING, and PROGRAM STATE sets. The first set is displayed as shown in Figure 64.

The COMMAND set of statistics data shown in Figure 64 indicates how many times each TRAMCON command, from UN to US, is entered at each terminal device defined on the given master. For example, in Figure 64, the value 143 alongside the PM command indicates that the PM command was entered 143 times at the system console or HIN terminal keyboard. These data are very useful for determining which TRAMCON commands are favored by the operators and which are of little or no use.

1. This master is in POLLER mode for segment UK2 because there are non-zero entries in the "Polls" column.
2. All remote units except **BFM** are being polled.
3. The remote unit just polled was **HYE** since it has been polled 100 times and the next remote unit has only been polled 99 times.
4. There is a physical break with the **CRS** remote unit or this Remote Unit is powered off because the unit has never responded.

```
      *** HIN ***         |      *** HIN1 ***
                          |
 UN    0    SI     1      |  UN    0    SI     0
 MA   75    DE   336      |  MA   35    DE   195
 SS  163    PR    33      |  SS   35    PR     1
 AL  300    LS    74      |  AL   24    LS     4
 AR   27    SC     0      |  AR   14    SC     1
 PA   30    SR     6      |  PA   10    SR     3
 ME   28    MS     4      |  ME   19    MS     1
 HE   13    OL     0      |  HE    7    OL     0
 HI    5    CO     2      |  HI    6    CO     0
 CN   15    ST     3      |  CN   12    ST     7
 PC    6    DI    66      |  PC    7    DI     7
 PH    0    LO     0      |  PH    8    LO     0
 SW   34    WH    16      |  SW    2    WH     2
 CR   19    LU     3      |  CR    8    LU     1
 CC   37    EQ     0      |  CC   10    EQ     0
 CF   10    UP     0      |  CF    0    UP     0
 PO   47    DN     0      |  PO    4    DN     0
 AC    0    OF     1      |  AC    0    OF     1
 IN    0    RU     0      |  IN    0    RU     0
 EN    1    VE     3      |  EN    0    VE     0
 DT  198    US     5      |  DT   45    US   131
 PM  143                  |  PM   21
 OP   12                  |  OP    0
 SE   20                  |  SE    1
 SM  215                  |  SM   60
```

Figure 64. US - TRAMCON operator command usage.

The data in Figure 65 show the TRANSMISSION status for every response received from each remote unit on each segment defined on the given master. The example in Figure 65 is for the Hillingdon, England, master, which has the two segments **UK2 and UK/BE** defined in its data base. From the example in Figure 65, one can conclude that the Hillingdon master is in MONITOR mode (NO entries in the "**Polls**" column) on segment UK/BE (UK - Belgium) and is NOT receiving any responses from any of the remote units on that segment. For segment **UK2** we can deduce the following:

*** UK2 ***						*** UK/BE ***					
	Polls	OK	PE	BR	NA		Polls	OK	PE	BR	NA
CRO	100	100	0	0	0	BFM	0	0	0	0	0
CRS	100	0	0	0	100	LDN	0	0	0	0	0
HYE	100	97	1	1	1	CDW	0	0	0	0	0
HYB	99	99	0	0	0	DNK	0	0	0	0	0
HIN	99	99	0	0	0	SWG	0	0	0	0	0
BFM	0	0	0	0	0	HOU	0	0	0	0	0
LDN	99	99	0	0	0	WEZ	0	0	0	0	0
						FLO	0	0	0	0	0
						CHE	0	0	0	0	0
						CH3	0	0	0	0	0
						SCC	0	0	0	0	0
						LEC	0	0	0	0	0
						FLR	0	0	0	0	0
						LEC	0	0	0	0	0
						SHR	0	0	0	0	0
						BNA	0	0	0	0	0
						SPP	0	0	0	0	0

Figure 65. US - TRAMCON segment transmission statistics.

The data in Figure 66 show a breakdown of the time required to process responses from each remote unit defined on each segment on the given master. As indicated by the notes, all values are in milliseconds and "----" indicates that a particular timing value is NOT being collected. For each remote unit, the time elapsed for the latest response (from when the poll message was issued to the update of the last display) is shown in the "**Total**" column. That "**Total**" time is further broken into three parts: disc I/O time (Disc), response transmission time (Xmit), and display update time (Disp). If bottlenecks in the remote unit response handling exist, they should be indicated here.

*** UK2 ***						*** UK/BE ***					
	Total	CPU	Disc	Xmit	Disp		Total	CPU	Disc	Xmit	Disp
CRO	----	0	0	----	0	BFM	0	0	0	----	0
CRS	----	0	0	----	0	LDN	0	0	0	----	0
HYE	----	0	0	----	0	CDW	0	0	0	----	0
HYB	----	0	0	----	0	DNK	0	0	0	----	0
HIN	----	0	0	----	0	SWG	0	0	0	----	0
BFM	----	0	0	----	0	HOU	0	0	0	----	0
LDN	----	0	0	----	0	WEZ	0	0	0	----	0
						FLO	0	0	0	----	0
						CHE	0	0	0	----	0
						CH3	0	0	0	----	0
						SCC	0	0	0	----	0
						LEC	0	0	0	----	0
						FLR	0	0	0	----	0
						LEC	0	0	0	----	0
						SHR	0	0	0	----	0
						BNA	0	0	0	----	0
						SPP	0	0	0	----	0

Notes: All values in milliseconds
----- indicates NOT being monitored

Figure 66. US - TRAMCON remote response timing.

The fourth page of US diagnostics is requested by entering "US,TS" and results in the execution of the program TS rather than US. Figure 67 shows the display produced when the operator enters "US,TS".

The PROGRAM STATE display shown in Figure 67 is divided into two parts. The top part gives a detailed account of how much of its lifetime a selected program spends in each possible PROGRAM STATE. Only the active programs that are selected by the operator are detailed in this portion. In the example, the two programs PLRP and MTRP were selected. Also from the example we can see that the program MTRP has spent its entire lifetime (100%) in the General WAIT (more specifically, the Class GET) state. Program PLRP has spent 1% of its time Scheduled to run, 2% of its time I/O Suspended, and the rest of the time in the Class GET General WAIT state.

```
 CPU:    0 %       |<------- WAITING -------->|<------- General  WAIT ------->
Prog  Dor- Sche-   |     for          Segmt|CL#  RN# LU/EQ buffr Class Lockd
Name  mant duled   | IO  Son Memry Disc  Swap |Allc Allc Down limit  get   LU
PLRP                 1    2                                               97
MTRP                                                                     100
--- PARTITION ---               -------------- PROGRAM ---------------------
Number Pages                    Priority Name    State

   1    32 RT R                     1    D.RTR
   2     5 BG                      99    RSM       Class GET         1
   3     5 BG                      10    AUTOR     I/O Wait          1
   4    12 BG                      99    UP26      I/O Wait          1
   5    18 BG                      53    TS25      <--Scheduled   Lock    1
   6    25 BG                      99    KYBRD     Class GET         1
   7    32 BG                      85    POLL      Class GET         1
   8    32 BG                      99    CMMD      Class GET      Swap    1
   9    49 BG                      99    PLRP      Class GET      Lock    1
  10    49 BG                      99    MTRP      Class GET      Lock    1
  11   190 BG SH SHAR1              9    $EMA$
```

Figure 67. US,TS - Real-time program state display.

The second portion of the display shows the real-time activity in each of the memory partitions. This display is ordered on the left by partition number. The partition number is followed by the partition size in 1024-word pages, the type (RT - Real-Time or BG - BackGround) and whether the partition is Reserved (R) and/or Sharable (SH). From the example, we can see that partition number 11 is the 190-page shared EMA partition called SHAR1. Partition number 1 is a 32-page Real-Time partition Reserved strictly for use by the disc I/O handler D.RTR. On the right side is the program Priority, program Name, and the current program State. The last two columns indicate whether the partition is Locked (Lock) by the resident program to prevent swapping and whether the present program is being Swapped (Swap). The information presented in this display is derived from either the Memory Address Table (MAT) or the ID segment for the particular program. The MAT is pointed at by the entry point $MATA and is described in detail in the RTE-6/VM Technical Specifications Manual (part no. 92084-90015) p. 8-39.

12.3 AL/AR Diagnostics

The diagnostic code in the module AL pertains mostly to the display of archive rather than Alarm/Status information. The archive system is very complicated and has been difficult to maintain without detailed diagnostics to aid the debugging process.

The first diagnostic information displayed as the operator enters the AR program is as follows:

```
heap^.archive_idx.arch_idx[i]

FRG1
     12      2    212    202    410    402    609    602    808    802
   1009   1002   1207   1202   1411   1402   1607   1602   1808   1802
   2024   2002   2208   2202   2408   2402   2605   2602   2805   2802
   3005   3002   3205   3202
FRG2
   3420   3436   3615   3623   3815   3823   4015   4008   4214   4208
   4416   4408   4614   4607   4814   4807   5014   5008   5214   5207
```

The example above indicates that there are 17 remote units defined for segment FRG1. There are 10 remote units defined for segment FRG2, with each pair of numbers corresponding to an individual remote unit in the order in which the remote units are defined in the data base. That is, the numbers 12 and 2 belong to the first remote unit defined on the first segment defined in the data base, FRG1. The numbers 3615 and 3623 belong to the second remote unit defined on segment FRG2 and so on.

These numbers are indices into the archive files (ARCH (REGULAR archive data) and (ARCHX (TRANSFERRED archive data) and are used to carefully maintain the integrity of the archive files. There are two values for each remote unit defined in the data base. Each pair of values are the next available record numbers in files (ARCH and (ARCHX for the remote unit represented by the array index "i". For example, the values 12 and 2 listed above are the next available record numbers in the files (ARCH (12) and (ARCHX (2) for the first remote unit in the segment FRG1. The records for each remote unit begin at intervals of 200 records starting with record number 2. Therefore, the numbers 12 and 2 indicate that there are 10 (12 - 2) valid records in file (ARCH, and 0 (2 - 2) records in file (ARCHX for the first remote unit in segment FRG1. In fact, the example above indicates that there are no TRANSFERRED archive records for any remote unit on segment FRG1.

The password-protected program CF is used by the experienced operator to initialize the archive files including the pointers being discussed here. This program does not check the exclusive use flag; therefore, the operator must ensure that all activity on the archive file is suspended before initializing any portion of this file.

The archive file uses the first record to store these NEXT AVAILABLE RECORD pointers and currently reserves 200(max_archive_record) records for each remote unit. Therefore, the record numbers that belong to the first remote unit are 2 through 201. The next 200 records, 202 through 401, belong to the next remote unit and so on. These are the respective ranges for each of these pointers discussed here and each set belonging to a particular remote unit is treated in a circular FIFO fashion. The corresponding pointers for the file (ARCHX are also kept in the first record of file (ARCH. This was done to avoid changing the definition of the HEAP. The pointers for file (ARCHX are offset by 62. That is, the two pointers for the first remote unit of the first segment defined are found in array locations 1 and 63. A copy of the first record on file (ARCH is kept in the HEAP array **"heap^.archive_idx.arch_idx"**. This array is used for quick update and could be used in the future to reduce access to the archive file if disc I/O activity needs to be reduced. Currently, each time an archive record is recorded, the index is updated in this HEAP array and record number one is updated on the disc from this array. The updating of record one could be done on a less often periodic basis, effectively halving the I/O involved in creating archive records. There are comments about this in the routine **"archive_it"** in library **$MPLIB**.

With these three sets of pointers displayed, there are three courses of action that can be taken. If the SPACE BAR is pressed, the program will re-list the pointers. This allows the operator to monitor the updating of these pointers in real-time as they are updated by any of the modules such as DT, MTRP, or PLRP. If the operator is dissatisfied with the values of these pointers, he may leave the AR program by pressing **"RETURN"**. The following messages will be displayed:

```
Archive file in use
TRY Again

f1: Enter Command
f2: DEB1 Status
```

If the operator is satisfied with the values of these pointers, he may proceed to view the archive records by pressing "z". The archive records will be displayed with diagnostic information interspersed with the actual archive data as follows:

```
 5/06:48:14    *** DEB1   Alarm/Status Archive ***
 HST - Hohenstadt                              FRC165 Pulsecom
recnum=  226  offset=   26

 5/06:48:14    *** DEB1   Alarm/Status Archive ***    25:    5/ 6:40:19
 HST - Hohenstadt                              FRC165 Pulsecom
ext_idx=   0
arch_idx=   25
nrecs=   25
nbr_extents=   0
arch_year0=-11989
arch_year1=    0
arch_year2=    0
max_category=    1  FALSE
next_archive_record=   227
ext_idx=   0  FALSE    -1
ext_idx=   0  FALSE     0
Hohenstadt link from Zugspitze             FRC-162/165
 MAJOR Receiver Baseband Degradation Red                   89/ 8:54:59 New
ext_idx=   0   TRUE     1
Hohenstadt link from Zugspitze             FRC-162/165
 MAJOR Receiver Baseband Degradation Red                   89/ 8:54:59 New
```

The diagnostic portion of the above display example is accented with **BOLD** and <u>**UNDERLINE**</u>. This diagnostic information pertains to the Multiple (extended) remote unit feature, discussed in Section 6.4. The physical record number is represented by "**recnum**" and is computed by the formula

recnum = (prevrem + r) * max_archive_record + offset , where

prevrem	= **the number of remote units in segments defined before the given segment. The segment order is derived from the "segment_info" array in the Data Base master record.**
r	= **the ordinal of the given remote unit for the given segment. The remote unit order is derived from the "remote_info" array in the Data Base SEGMENT record.**
offset	= **the number of records from the BASE record for the given remote unit. The BASE record number for any given remote unit is the lowest value of the range of values discussed above.**

In the example above, the given segment is DEB1 and is the first segment (ordinal 0) defined in the master record. This means "**prevrem**" equals 0. The given remote unit is Hohenstadt, which is the second remote unit (ordinal 1) defined in the DEB1 segment record. This ordinal is the value of "**r**". The constant "**max_archive_record**" is currently set to 200. The BASE record number for the Hohenstadt remote unit on segment DEB1 is (0 + 1) * 200. Since disc file record numbers begin with 1, we must add 1, to the BASE value of 200. Also, as explained above, we must add 1 because the first record of this file is used to store the index. So the ABSOLUTE BASE record number for the Hohenstadt remote unit is 202. IF there are fewer than "**max_archive_record**" records on file for Hohenstadt, then the RELATIVE BASE record number equals the ABSOLUTE BASE value. Otherwise, the RELATIVE BASE number equals one record number beyond the NEXT AVAILABLE RECORD number indicated by the pointers above. The NEXT AVAILABLE RECORD number for Hohenstadt must be a value in the range 202 - 401.

In the example, "**nrecs**" indicates that there are only 25 archive records for this remote unit. Therefore, the LOGICAL BASE equals 202.

The "**offset**" equals the number of records that the record being displayed is removed from the LOGICAL BASE RECORD number. The logical record being displayed is shown in the upper right-hand corner of the display as the first value in the time/date stamp (in the example, the logical record number is 25). This logical record number is the value of local variable "**arch_idx**". Therefore, the actual physical record number of the archive record being displayed is 226 and is represented by "**recnum**".

The GLOBAL VARs "**ext_idx**" and "**nbr_extents**" are used to track MULTIPLE Remotes and is explained in Section 6.4. In our example, "**nbr_remotes**" equals zero indicating that the Hohenstadt remote unit is a single unit.

The year value of each archive record indicates that the record was created to mark a NO ANSWER event by being set to the negative of the actual year value minus 10000. This value is displayed as "**arch_year0**" and in the example, the actual year is 1989.

12.4 CR Diagnostics

The program CR is a password-protected program and is therefore NOT covered in the TRAMCON Operator's manual. Nevertheless, this program is and will continue to be a source of very valuable diagnostic information concerning the operational status of each of the terminal devices defined on a given master. The first page of this CR display, shown below, is a summary of all the terminals defined on this master.

```
218/14:46:21           *** CRT Status ***          Opr: Rick Statz
# Location             CurSeg   Current Display    Operator Name   Default Disp
0 Schoenfeld           DEB4C:M  CRT Information    Rick Statz      Seg Status
1 Schoenfeld2          DEB4C:M  CRT is DOWN        Logged OFF      Segment Map
```

A second page, shown below, presents more detail for individually selected terminal devices. The first line of this display is the normal nondiagnostic portion. The information starting with "**line_nbr**" is the data available through the diagnostic function.

```
218/14:46:21  *** Schoenfeld     CRT (0) ***                Opr: Rick Statz
Gr Mode   Terminal                            Parity  Hard/      Baud  Auto
Graphics  Type      Printer  Parity Color     Sense   Modem      Rate  Answer
F / T     HP-2397A  HP-2932A  OFF   TRUE       ODD    Hardwire   9600  FALSE

                              first_line   lines
       line_nbr:      0           0          0
      nbr_lines:      0           0          0
  pgs_remaining:    100           0          0
       cur_page:      0           0          0
     prev_pages:      0           0          0
       max_page:      5           0          0
        sav_len:     -1
           misc:      0           0     1    1 11788    10
      locked_ln:      0
  sav_fkey_entr:      5
     fkey_entry:      5
     max_dsp_ln:     23
        max_key:      4
        sav_dsp: SS
        old_dsp: CR
  remotes_displ: []
```

The information from "**line_nbr**" to "**remotes_displ**" is kept in the HEAP record "**current_crt**" for each terminal defined in the Configuration data base for the given master. This is a good place to look for a problem if a terminal is failing to respond or interact with an operator.

12.5 DT Diagnostics

Since the original version of the DT program was fielded, many improvements have been made to make this very complicated program more user-friendly and more intelligent. One of the greatest problems was the fact that the operator could be led quite far into the DT process before learning that the desired data transfer process was not possible at this time. This information is now available at the start of the DT process and has been made known early-on to the operator.

For example, when the operator is presented with the list of TRAMCON masters, as shown in Figure 68, some or all of those masters may not be communicating with this master at this time. The information is conveyed to the operator by displaying in red the masters that cannot be reached, and in green, the ones that can be reached.

```
        List of masters in the TRAMCON Network. This node is Vaihingen
 What other master should be used for Data Transfer?

           A. Enter Command                  Green = Master Available
           B. Default Display                Red   = Master NOT Available

           1. (DON ) Donnersberg             15. HDG  Heidelberg
           2. (RAG ) Reese-Augsburg          16. FKT  Frankfurt
           3. (FEL ) Feldberg                17. HAN  Hahn
           4. (BLN ) Berlin                  18. RSN  Ramstein
           5. (GAR ) Garlstedt
           6. (CHE ) Chievres
           7. (SCH ) Schoenfeld
           8. (KKR ) Kalkar
           9. (CRO ) Croughton
          10. (HIN ) Hillingdon
          11. (DBG ) Nuernberg
          12. (AVO ) Aviano
          13. (CLO ) Coltano
          14. (SGT ) Stuttgart

        Enter NODE # or SITE CODE:
```

Figure 68. DT - List of masters display.

In Figure 68, the operator is reminded in the upper right-hand corner that his/her machine is the Vaihingen master and, of course, Vaihingen does not appear in the list because a given master cannot transfer data to/from itself. Also in the example, the three masters with which communication has been established at this time are underlined. This means that, at the present time, the Vaihingen master can physically contact the Schoenfeld, Aviano, and Coltano masters.

The number of masters listed in display shown in Figure 68 is dependent upon the master password (refer to section 12.3). If the master password was entered to set "**access_restricted**" to FALSE, all the masters in the network are displayed. If "**access_restricted**" is TRUE, only the immediate neighbors are listed.

Just because a master is shown in green does NOT necessarily mean that the Vaihingen operator wishes to or CAN exchange data with that master. A particular master can (this won't happen often) be operational and have a working communication channel to the contacting master while it is running the TRAMCON software. In most cases, this would mean that the DT session is not possible with the particular master because the operator is usually interested in transferring TRAMCON data which requires that the TRAMCON software be running on both machines. This piece of information is added to the display in Figure 68, as shown in Figure 69, if the "**diag**" flag is set.

Two columns of information have been added to the display in Figure 68 resulting in the display shown in Figure 69. The first column contains a BLANK or an "**N**" indicating that the corresponding master is a Neighbor of Vaihingen. That is, the Aviano and Coltano masters are directly connected to the Vaihingen master and have no other TRAMCON masters as intermediate nodes.

```
          List of masters in the TRAMCON Network.  This node is Vaihingen
     What other master should be used for Data Transfer?

            A. Enter Command                  Green = Master Available
            B. Default Display                Red   = Master NOT Available

  D         1. (DON ) Donnersberg             15. HDG  Heidelberg
  D         2. (RAG ) Reese-Augsburg          16. FKT  Frankfurt
  D         3. (FEL ) Feldberg                17. HAN  Hahn
  D         4. (BLN ) Berlin                  18. RSN  Ramstein
  D         5. (GAR ) Garlstedt
  D         6. (CHE ) Chievres
  U         7. (SCH ) Schoenfeld
  D         8. (KKR ) Kalkar
  D         9. (CRO ) Croughton
  D        10. (HIN ) Hillingdon
  D        11. (DBG ) Nuernberg
N U        12. (AVO ) Aviano
N D        13. (CLO ) Coltano
  D        14. (SGT ) Stuttgart

          Enter NODE # or SITE CODE:
```

Figure 69. DT - List of masters display with diagnostics.

The second column of diagnostic information contains either the letter "D" or the letter "U" indicating that the TRAMCON software is either DOWN (not running) or UP (running) at the corresponding master. In Figure 69, the only masters that have the TRAMCON software operational are Schoenfeld and Aviano.

The display shown in Figure 70 would be the result of successfully contacting the Schoenfeld master. The information highlighted in the middle of the display is a result of having the "**diag**" flag set. The data displayed verify that the Schoenfeld master was contacted. The response should always be 47 words long. The information received is read from the (DATE file at the far end (Schoenfeld in this case). From the diagnostic data in Figure 70, the user can tell that 47 words were returned to Vaihingen. Those 47 words inform the DT operator that the TRAMCON software is operational (TRAMCON: UP) at Schoenfeld and that the TRAMCON Configuration Data at Schoenfeld has two Segments (DEB4C and DEB3S) defined with 10 remote units and 13 remote units, respectively.

The message "**The two masters have no TRAMCON data in common.**" appears with or without diagnostics set and indicates that the two masters (in this case Vaihingen and Schoenfeld) do not monitor any common TRAMCON segment. In this example, Vaihingen monitors the DEB1 segment and Schoenfeld monitors the DEB4C and DEB3S segments. Therefore, the operator is not offered the option of transferring TRAMCON data such as Archive records or Calibration Curve data. This is the look-ahead, time-saving code that has been added so that the operator will not be led deep into the DT process just to discover that there is no information to be shared with the chosen master.

```
          Calling Schoenfeld
          Please WAIT

     {    47 words read, 27435 TRAMCON: UP            }
     {    0. DEB4C      10                            }
     {    1. DEB3S      13                            }
     {    2. Undefined                                }
     {    3. Undefined                                }

          The two masters have no
          TRAMCON data in common.

          Press RETURN to continue.
```

Figure 70. DT - Contact master display with diagnostics.

Figure 71 is an example of the display presented by DT when data base information is actually exchanged between masters. The example in Figure 71 indicates that TRAMCON alarm/status archive records were transmitted to the master at Schoenfeld.

```
          Transmitting Archives to Schoenfeld

         Remote Unit: Schoenfeld
              Segment: FRG2

              37 Blocks ( 37 recs) to transmit
              37 Blocks transmitted
  {              from:  3424  3460                     }
  {                to:     2    38                     }
  {       new pointer:    39                           }

         Data Transfer Completed Successfully

  {       Transfer Time: 1 min  57 secs                }

  {       9472 bytes @ 80 bytes/sec                    }
  {       37 records (125 wds/rec)                     }

          Press RETURN to continue.
```

Figure 71. DT - Data transmission display with diagnostics.

The archive records transmitted belonged to the Schoenfeld Remote Unit on segment FRG2. There were 37 valid records to transmit, of which 37 records were successfully transmitted. The information marked with {} is presented to the operator if the "**access_restricted**" is FALSE and the system diagnostic flag "**heap.diag**" is TRUE. The first three lines of diagnostic information indicate the starting and ending record numbers at the transmitting (from) master and the starting and ending record numbers at the receiving (to) master. In the example, the archive records from 3424 thru 3460 were transferred to record numbers 2 thru 38 at the Schoenfeld (receiving) master. The third line is shown for archive data only. It represents the "**next_archive_record**" pointer which is changed at the receiving master to reflect the new amount of archive information for the given Remote Unit. The last three lines of diagnostic information informs the operator of the transfer rate. This information should be checked if transfer times seem excessive.

12.6 SE Diagnostics

The program SE displays status information for each of the TRAMCON segments defined in the Configuration data base for a given master. Shown below is the display produced by the SE program.

```
225/22:05:37  *********** Segment  Information ***********
     Short                                           TMT I/O
     Segment                                         Channel  Number
Ord  Name          Long  Segment  Name       Status  (Octal)  remotes
 0.  UK2      United Kingdom Segment 2       Poller    15        7
 1.  UK/BE    United Kingdom / Belgium       Monitor   16       17
```

In the sample SE display above, the master has two segments, UK2 and UK/BE, defined in its Configuration data base. These segments can be referred to in TRAMCON command entry by using either the "**Short Segment Name**" or the corresponding ordinal "**Ord**". Below is the SE display as it appears when the "**diag**" flag is enabled.

```
225/22:05:37  *********** Segment  Information ***********
     Short                                           TMT I/O
     Segment  Shared Memory (EMA,wrds decimal)       Channel  Number
Ord  Name     Start    Stop    Required      Status  (Octal)  remotes
 0.  UK2      141376   117573   23804        Poller    15        7
 1.  UK/BE    117573    57903   59670        Monitor   16       17
```

The Long Segment Name has been replaced by some shared memory (EMA) information for each segment defined. These values are the HEAP variables EMA_start, EMA_end and EMA_required, respectively. The meaning of these values is discussed in Section 11.1. In this example, segment UK2 has seven remote units and requires 23804 words of EMA storage, and segment UK/BE has 17 remote units and requires 59670 words of EMA. The "**stop**" value for the last defined segment (57903 for UK/BE above) indicates the amount of EMA currently unused.

13. SOFTWARE DISTRIBUTION AND SYSTEM RECOVERY FROM DISC FAILURE

The software distribution procedure became relatively simple once the new 7912 disc drive was incorporated into the TRAMCON system. The software has always been distributed as a bit-for-bit copy of the disc. With the old 7906 disc drive, only the portion of the disc containing the software could be saved and restored. This partial disc save-restore is no longer an option on the new 7912 drive. The entire disc must be saved and restored, but the procedure for saving and restoring has been reduced to the pushing of a few buttons rather than the lengthy interactive save-restore routines that were previously used. The major drawback to saving the entire disc is the fact that the Run-time data files, such as the archive file (ARCH and the calibration curve file (CC, are also overwritten with the contents of these files from the software distributor's machine. If software is being distributed to a newly-installed machine, the distributor should ensure that these data files are initialized to the default values by running the program CF for all files and segments. If new software is being sent to an operational machine, these files should be saved with the File Copy (FC) routine before the new software is installed and restored with the FC routine after the new software is operational. Files that one might want to save include:

1. "HE - The procedures HELP file if any local modifications,
   "HEIDX such as to the SOP, were made using the ED command
2. (ARCH - Alarm/status archives
3. (CC - Parameter calibration curves and thresholds
4. All three sets of Configuration data base files

NOTE: These files ALL reside on disc LU 10 and have the exclusive security code 2810 so that they can be easily saved and restored as a set by specifying all files that fit the descriptor "------:10:2810".

Current	New	Backup
(DICT	)DICT	^DICT
(NET	)NET	^NET
(LINKS	)LINKS	^LINKS
(MAST	)MAST	^MAST
(LINK	)LINK	^LINK
(REMOT	)REMOT	^REMOT
(SEG	)SEG	^SEG
(TRUNK	)TRUNK	^TRUNK
(EQT	)EQT	^EQT
(CRT	)CRT	^CRT
(SITE	)SITE	^SITE
(DINIT	)DINIT	^DINIT

The save-restore procedure for the new disc drive is referred to here as a PUSHBUTTON save or restore. The pushbutton-save copies the entire contents of the 7912 disc including formatting information to a 600-foot pre-formatted cassette tape. The pushbutton-restore copies the entire contents of the cassette tape to the 7912 disc. Both of these functions are performed entirely off-line by the disc unit and require no support from the HP-1000.

1. STOP all activity on the TRAMCON master.
2. Remove the front panel from the 7912 disc drive by using the fingerholds on either side of panel and pulling straight out.
3. Place a formatted and certified 600-foot cassette tape in the drive located on the front of the disc drive. Tapes may be purchased from HP already certified and ready to use for approximately $37 per cassette. They may also be purchased directly from 3M already formatted for the HP-7912 (Model DC600HC) but uncertified for approximately $17 per cassette. The cassettes can be certified using the On-Line routine FORMC. This certification takes approximately 1 hour per cassette. Allow approximately 3 minutes for the tape to load. The tape drive makes an unusual clacking sound when the tape is loaded and the busy light on the tape drive will turn off. The busy light is the yellow LED on the left and the write protect is the yellow LED on the right. Make sure that the write-protect tab located in the upper left hand corner of the cassette points away from the SAFE setting. If this is set properly, the write-protect LED will NOT be lit.
4. **This is the crucial step.** With a simple button press, an entire medium, cassette tape or 7912 disc, will be irreversibly overwritten. First, locate the red save-restore switches on the front of the disc drive immediately below the cassette tape drive. These switches are labeled to indicate their function. One switch (S1) is for the save operation, which copies data FROM the disc drive ONTO the cassette tape, destroying the previous contents of the cassette tape. The other switch (S2) is used to restore the data FROM the cassette tape ONTO the disc drive, destroying the previous contents of the disc. As a precaution, the switches are labeled as follows:

S2	**S1**
TAPE ---> DISC	**DISC ---> TAPE**
RESTORE	**SAVE**

   The diagram above indicates that switch S2 is for the restore or tape-to-disc procedure and switch S1 is for the save or disc-to-tape procedure. The switches are momentary toggle switches that must be pushed to the right to activate. They will automatically return to the left rest position. As an added precaution, to perform the save or restore function the appropriate switch must be pushed to the right, which will cause the busy LED indicator to blink for approximately 8 seconds. If the same switch is NOT pressed a second time within the 8 seconds, the operation will be aborted. If the switch **is** pressed a second time within 8 seconds, the save or restore operation will proceed for approximately 25 minutes, saving or restoring the entire 65 Mbytes.
5. After the restore procedure is performed, the system should be booted-up by following the procedure detailed in Section 15 of this manual to activate the new system software just restored to the disc.

Figure 72. Pushbutton-save or restore procedure.

In summary, the software distribution procedure is as follows:

1. Software maintenance organization records the software version number and date/time-stamp in file (DATE by executing the program SETVE after insuring that the development system clock is set to the current time/date. The program SETVE is executed by entering the program name as an FMGR command and specifying the version number as the first and only run-time parameter as follows: "SETVE,1.8" to set the version number to 1.8. The SETVE program reads the system time/date clock in the two-word integer format (seconds since 00:00 1 January 1970) and places the time/date read into the two-word integer variable "version_date" in the "date_record" in disc file (DATE. The version number is read as a real number and placed in real variable "version_nbr" in record "date_record" in disc file (DATE. This version number and time/date-stamp are displayed by routine "logo" in program INIT when the TRAMCON system is booted-up and by routine "simple_cmds" in program CMMD upon operator request via the VE command.

2. Software maintenance organization performs the save function described above and mails the cassette to the field.

3. Site personnel save all disc files that contain information specific to the site as discussed above.

4. Site personnel perform the restore procedure as described above.

5. Site personnel restore the disc files saved in Step 3 using the FC program as described above.

The software distribution tape should be kept by the site personnel indefinitely and used to recover the system any time there is a disc failure. Whenever the data on the disc are lost by hardware failure or diagnostic maintenance routines that overwrite the disc, Steps 4 and 5 must be performed. Step 3 should be done periodically so that the data restored in Step 5 will be as current as possible.

14. CONFIGURATION DATA BASE DISTRIBUTION AND IMPLEMENTATION

As stated in the introduction to this manual, the TRAMCON software was written to be as independent of the environment as possible. In order to accomplish this, the TRAMCON On-Line software relies on the Configuration data base for any specific information about the environment in which it operates for a given master. This section describes how changes made to the Configuration data base are distributed to field sites and subsequently introduced to the operational TRAMCON system at those sites. The data base creation and distribution process is depicted in Figure 73. A program called the Configurator maintains the data base for the entire TRAMCON network. At any time, the Configurator can generate a subset of this overall data base for any given TRAMCON master system. These subset data bases are referred to as master-specific data bases and consist of the 12 disc files listed in Figure 73.

A new data base consisting of the 12 files whose names begin with ")" is distributed to field on tape cassette in File Copy (FC) format. The RE procedure file shown in Figure 75 is required to be on disc LU 10 of every TRAMCON field system and is used by the field personnel to copy these new files from tape to disc. The statement "**:RU,FC,CO,-8,,BDV**", runs the program FC and instructs it to copy the entire contents of the tape (LU = 8) to disc, replacing any files of the same name (D) and verifying the results (V).

15. SYSTEM POWER FAILURE AUTORECOVERY AND SYSTEM BOOT-UP

This section discusses the general concepts of system autorecovery from power failure and system boot-up. The mechanics of the boot-up procedure are carefully detailed both On-Line under the HE,BO command and on hard copy in the TRAMCON User's Manual, p. 76. The boot-up procedure is intended to be used sparingly in the field to recover from a failure that has caused the TRAMCON programs to stop functioning. In the operational system, a failure of this severity should be rare. As stated in the introduction, the TRAMCON master system is designed with battery reinforced memory and time/date clock to survive power failures anywhere from short fluctuations up to 3 hours of constant power outage. To "survive" a power failure means that business resumes as usual after the power failure and the only indication that the power failure occurred is the record of the event kept by the power fail recovery program AUTOR. This power failure recovery system is diagrammed in Figure 76.

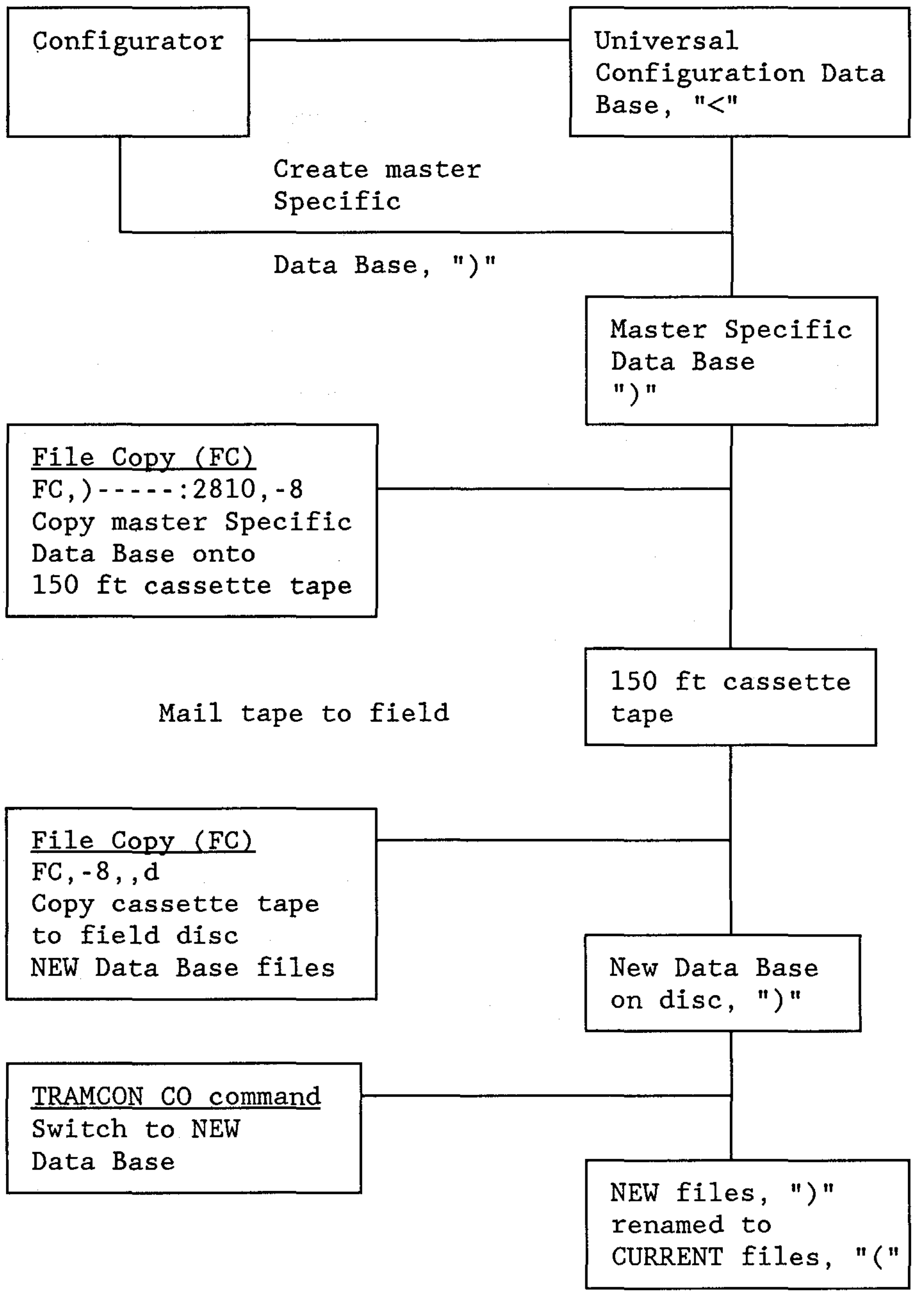

Configurator
Universal Configuration Data Base, "<"
Create master Specific
Data Base, ")"
Master Specific Data Base ")"
File Copy (FC)
FC,)-----:2810,-8
Copy master Specific Data Base onto 150 ft cassette tape
Mail tape to field
150 ft cassette tape
File Copy (FC)
FC,-8,,d
Copy cassette tape to field disc NEW Data Base files
New Data Base on disc, ")"
TRAMCON CO command
Switch to NEW Data Base
NEW files, ")" renamed to CURRENT files, "("

Figure 73. Configuration data base creation and distribution.

	File Name	Number Records	Record Size(words)	Description
1.	)DICT	1	7000	Dictionary
2.	)NET	1	140	IPC Network
3.	)LINKS	1	1750	Links defined in Network
4.	)MAST	1	47	TRAMCON master record
5.	)LINK	indef	119	Link-End records
6.	)REMOT	indef	15	remote unit records
7.	)SEG	indef	244	Segment records
8.	)TRUNK	indef	76	Digroup trunk records
9.	)EQT	indef	2313	Transmission equipment records
10.	)CRT	1 to 5	6	Terminal (CRT) records
11.	)SITE	indef	9	Site records
12.	)DINIT	TEXT	variable	DS Initialization (for DINIT)

Figure 74. Master specific configuration data base files.

```
:SV,4,,IH
:TE, ***********************************************
:TE, ***  Installing NEW Configuration Data      ***
:TE, ***********************************************
:RU,FC,CO,-8,,BDV
::)MISC
```

Figure 75. RE - New configuration data base REplacement.

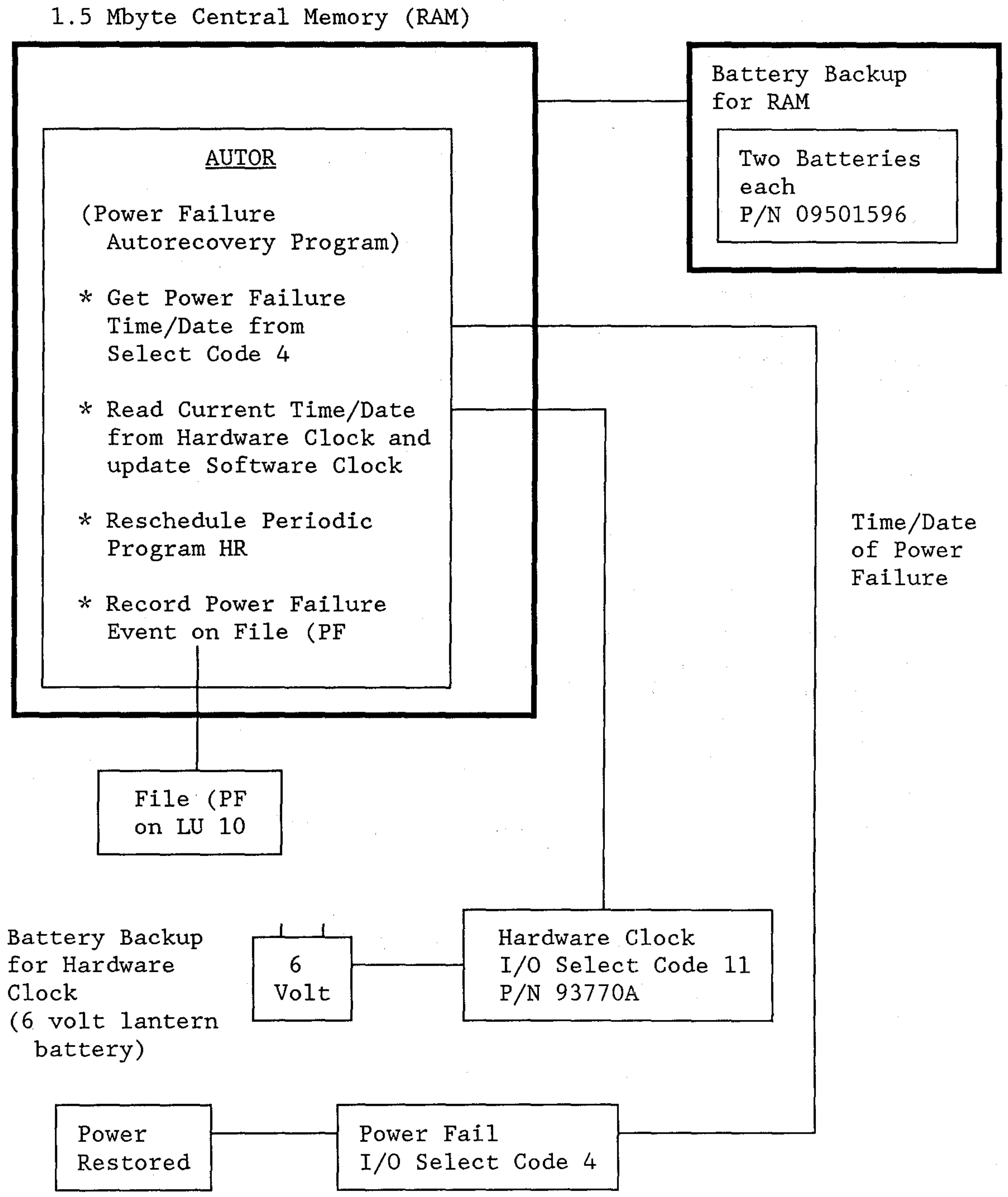

1.5 Mbyte Central Memory (RAM)
AUTOR
(Power Failure Autorecovery Program)
* Get Power Failure Time/Date from Select Code 4
* Read Current Time/Date from Hardware Clock and update Software Clock
* Reschedule Periodic Program HR
* Record Power Failure Event on File (PF
Battery Backup for RAM
Two Batteries each P/N 09501596
Time/Date of Power Failure
File (PF on LU 10
Battery Backup for Hardware Clock (6 volt lantern battery)
6 Volt
Hardware Clock I/O Select Code 11 P/N 93770A
Power Restored
Power Fail I/O Select Code 4

Figure 76. Power failure automatic recovery system.

16. REFERENCES

Farrow, J.E. and Skerjanec, R.E. (1986), Transmission Monitoring and Control of Strategic Communication Systems, IEEE Journal on Selected Areas in Communications, SAC-4, No. 2, March 1986.

Hewlett-Packard Company (1981), RTE-6/VM On-Line Generator Reference Manual, Part No. 92084-90010.

Hewlett-Packard Company (1981), RTE-6/VM System Manager's Reference Manual, Part No. 92084-90009.

Hewlett-Packard Company (1981), RTE-6/VM Technical Specifications Manual, Part No. 92084-90015.

Hewlett-Packard Company (1981), EDIT/1000 User's Manual, Part No. 92074-90001.

Hewlett-Packard Company (1981), Pascal/1000 Reference Manual, Part No. 92833-90001.

Hewlett-Packard Company (1981), Fortran 77 Reference Manual, Part No. 92836-90001.

Hewlett-Packard Company (1981), Macro/1000 Reference Manual, Part No. 92059-90001.

Hewlett-Packard Company (1981), Relocatable Library Reference Manual, Part No. 92084-90013.

Hewlett-Packard Company (1981), RTE-6/VM Loader Reference Manual, Part No. 92084-90008.

Hewlett-Packard Company (1981), RTE-6/VM LINK User's Manual, Part No. 92084-90038.

Hewlett-Packard Company (1981), RTE-6/VM Utility Programs Reference Manual, Part No. 92084-90007.

Hewlett-Packard Company (1981), RTE-6/VM Programer's Reference Manual, Part No. 92084-90005.

Hewlett-Packard Company (1981), RTE-6/VM Quick Reference Guide, Part No. 92084-90003.

Hewlett-Packard Company (1984), HP-2647F Reference Manual, Part No. 02647-90037.

Hewlett-Packard Company (1985), HP-2627A Reference Manual, Part No. 02627-90002.

Hewlett-Packard Company (1986), HP-2397A Reference Manual, Part No. 02397-90002.

APPENDIX A: PROCEDURE FILES FOR IMPLEMENTING PROGRAM "CONFI"

Compiling, Indexing, Segmenting, Loading, and Saving Program CONFI

This appendix lists all the FMGR procedure files used to implement changes made to the data base Configurator program CONFI. The first file listed in Figure A-1 purges the previous relocatable files for all segments of program CONFI and packs the disc cartridge on which they reside.

```
:*******************************************************************
:*** FMGR Procedure File - RUNCL  Purges Relocs & Compiles  CONFI ***
:***                                                             ***
:*** This Procedure File does the following:                     ***
:***   1. Purges the old Relocatable files for the program CONFI ***
:***   2. Compiles all Segments of program CONFI                 ***
:***   3. Transfers to Procedure File RUNC which Indexes,        ***
:***      Segments, Loads and Saves the program CONFI.           ***
:*******************************************************************
:PU,%CONS0::10
:PU,%CONS1::10
:PU,%CONS2::10
:PU,%CONS3::10
:PU,%CONS4::10
:PU,%CONS5::10
:PU,%CONS6::10
:PU,%CONS7::10
:PU,%CONS8::10
:PU,%CONS9::10
:PU,%CONSA::10
:PU,%CONFI::10
:PK,10
:RU,P,&CONS0,,%CONS0::10:5:120
:RU,P,&CONS1,,%CONS1::10:5:120
:RU,P,&CONS2,,%CONS2::10:5:120
:RU,P,&CONS3,,%CONS3::10:5:120
:RU,P,&CONS4,,%CONS4::10:5:120
:RU,P,&CONS5,,%CONS5::10:5:120
:RU,P,&CONS6,,%CONS6::10:5:120
:RU,P,&CONS7,,%CONS7::10:5:120
:RU,P,&CONS8,,%CONS8::10:5:120
:RU,P,&CONS9,,%CONS9::10:5:120
:RU,P,&CONSA,,%CONSA::10:5:120
:RU,P,&CONFI,,%CONFI::10:5:120
:TR,RUNC::10
::
```

Figure A-1. FMGR procedure file for compiling CONFI - RUNCL.

The procedure file in Figure A-1 places the relocatable modules for the program CONFI onto disc LU 10. At this point the program CONFI consists of 11 segments and the main program. Once all the modules for CONFI are compiled, the RUNCL procedure file transfers to the procedure file RUNC, which indexes, segments, edits, loads, and saves program CONFI. FMGR procedure file RUNC is listed in Figure A-2.

Implementing the program CONFI is a lengthy process. The clock is displayed using the **RTE TI** command at the start of major steps to inform the programer how long each step has been running. The **RTE TI** command is issued three times with the FMGR command "SYTI" as shown in Figure A-2. The first step in procedure RUNC is to gather together all the relocatable modules used in program CONFI and index them. This indexing step is performed by program INDXR, which gets its instructions from file #RUNCL and produces the indexed file of relocatable modules called %CONF. The INDXR directive file #RUNCL is listed in Figure A-3.

```
:*******************************************************************
:*** FMGR Procedure File - RUNC   Indexes, Segments & Loads CONFI ***
:***                                                             ***
:*** The program CONFI is assumed to be Compiled before this     ***
:*** Procedure File is executed.                                 ***
:***                                                             ***
:*** This Procedure File does the following:                     ***
:***    1. Indexes the CONFI relocatables using program INDXR    ***
:***    2. Segments CONFI producing loader directive file #CONFI, ***
:***    3. Edits the comments and spaces out of file #CONFI,      ***
:***    4. Loads the program CONFI with MLLDR and                ***
:***    5. Saves the program on LU 10                            ***
:*******************************************************************
:SYTI
:PU,%CONF::10
:RU,INDXR,#RUNCL::10
:PU,#CONFI::10
:RU,SGMTR,%CONF::10,#CONFI::10:4:100,29,CONFI,D
:PU,#Z::10
:RU,EDIT,#CONFI::10,TR,^RUNCL::10/
:PU,#CONFI::10
:ST,#Z::10,#CONFI::10:4:-1
:OF,CONFI
:SYTI
:RU,MLLDR,#CONFI::10
:PU,CONFI::10
:PK,10
:SP,CONFI::10
:SYTI
:EX
```

Figure A-2. File for indexing, segmenting, and loading CONFI - RUNC.

```
CR,%CONF::10:5:200
IN,%CONFI::10
IN,%CONS0::10
IN,%CONS1::10
IN,%CONS2::10
IN,%CONS3::10
IN,%CONS4::10
IN,%CONS5::10
IN,%CONS6::10
IN,%CONS7::10
IN,%CONS8::10
IN,%CONS9::10
IN,%CONSA::10
IN,%CNLIB::10
IN,$PLIB
IN,%PLDH2
IN,$SHSLD
IN,$FMP6
EN
```

Figure A-3. INDXR directive file for program CONFI - #RUNCL.

File #RUNCL listed in Figure A-3 directs program INDXR to create an indexed library file called %CONF on disc LU 10 and include it in the main program relocatable for program CONFI, the 11 segment relocatable modules for program CONFI, a library of routines for program CONFI in file %CNLIB, the Pascal library $PLIB, the short EMA routines in file %PLDH2, the short run-time error message routines in file $SHSLD, and the library $FMP6.

The output of the INDXR step shown in Figure A-3 is first used by program SGMTR to automatically segment program CONFI. As shown in Figure A-2, the segmenter SGMTR is directed to segment the modules it finds in file %CONF and produce a loader directive file called #CONFI on disc LU 10. The main program is called CONFI, the path length is limited to 29 pages and the segments are to be disc (D) resident.

The output produced by the segmenter as stated above is disc file #CONFI. This file is filled by program SGMTR with unusable comments and spaces. Removing these comments and spaces can speed-up the loading process considerably (see Section 8.2.3 of this manual). The removal of this unwanted text is accomplished using the program EDIT. As shown in Figure A-2, the program EDIT is directed to edit file #CONFI and look in file ^RUNCL for editing instructions. The editor instruction file ^RUNCL is listed in Figure A-4.

```
f/TOTAL PROGRAM SIZE/|/|k|-1|j|p=|g/ /_/|1|j|3| LI,%PLDH2| LI,$SHSLB
sere on|3$x/ //q|3$d/^[A-Z,=]/aq|1|sewcl,1|f/=/|sewc|p*|g/_/ /|ec#Z::10
```

Figure A-4. EDIT instructions for CONFI segmentation file - ^RUNCL.

A detailed discussion on how these EDIT command files are interpreted and used can be found in Section 8.2.3 of this manual. Basically, file ^RUNCL in Figure A-4 instructs program EDIT to remove all unnecessary text such as comments and spaces from the file specified in the edit run-string (in this case, file #CONFI). The two library directives for the loader, "LI,%PLDH2" and "LI,$SHSLB", are inserted after line 3. The edited file is placed on the temporary file #Z so that the excess file space created by this editing step, #CONFI, can be recovered.

APPENDIX B: GENERATION ANSWER FILE "ANTR"

```
*********************************************************************
*   Answer File ANTR for TRAMCON System    last edited <871021.1250>  *
*                                                                     *
*   System has following I/O Slot (Select Code) Configuration:        *
*                                                                     *
*   Select Code                  Device - Interface                   *
*   -----------   ----------------------------------------------------*
*        4        Power Fail                                          *
*       10        FEM board for VM 1000 firmware                      *
*       11        Time/Date Clock (                                   *
*       12        7912 System Disc (12821A interface)                 *
*       13        2397A System Console (BACI 12966A)                  *
*       14        2647F,2627A or 2397A terminal (BACI 12966A)         *
*       15        TRAMCON segment 1 (BACI 12966A)                     *
*       16        TRAMCON segment 2 (BACI 12966A)                     *
*       17        2647F,2627A or 2397A terminal (BACI 12966A)         *
*       20        DS channel 1 (12794B - p/n 5061-4913)               *
*       21        DS channel 2 (12794B - p/n 5061-4913)               *
*       22        Relay Output (                                      *
*       23        Unused                                              *
*       24        Unused                                              *
*       25        2647F,2627A or 2397A terminal (BACI)                *
*********************************************************************
"TRMCN::10                      * Generation List file name
YES                             * Echo questions and answers
!TRMCN::10::5000                * Generation Output file name
7912                            * System disc type: CS80, 65 Mbytes
12                              * System disc Select Code (I/O Slot)
*********************************************************************
*   Disc Subchannel definitions, 7912 disc layout                     *
*********************************************************************
* model: CTD
* HP-IB: 0
* unit:  1
* volume: 0
* initial number of blocks: n/a
*
* subchannel#/
* disc cache          n/a
* ----------
* 0 assigned          n/a
*
* model: 7912
* HP-IB address: 0
* unit: 0
```

Figure B-1. TRAMCON field system generation answer file - ANTR.

```
* volume: 0
* initial number of blocks: 256,256
*
* subchannel#/                                         blocks     blocks
* disc cache       # of tracks   blocks/track   extended   remaining
* -------------   -----------   ------------   --------   ---------
* 1 (system)          1000            64          64000      192256
* 2 (data)            3000            64         192000         256
* disc cache           CTD             0            256           0
* disc input format = device(model,hp-ib address,unit,volume)
CTD,0,1,0                     * integrated cartridge tape drive
7912,0,0,0                    * disc definition
1000,64                       * subchannel 1 (system + all progs)
3000,64                       * subchannel 2 (TRAMCON data files)
CTD,0                         * disc cache
/E                            * TERMINATE SUBCHANNEL DEFINITION
**********************************************************************
1                             * System subchannel
NO                            * AUXILIARY DISC?
11                            * TBG SELECT CODE
0                             * PRIV. INT. SELECT CODE
YES                           * MEM. RES. ACCESS TABLE AREA II
YES                           * RT MEMORY LOCK
YES                           * BG MEMORY LOCK
50                            * SWAP DELAY
512                           * Memory Size (1024 word pages)
0                             * NO boot file
**********************************************************************
*      RELOCATABLE MODULES                                           *
**********************************************************************
*                 RTE-6/VM OPERATING SYSTEM
**********************************************************************
MAP MODULES, LINKS
LINKS IN CURRENT
REL,%CR6S1
REL,%CR6S2
REL,%CR6S3
REL,%$CNFG
**********************************************************************
*                 I/O DRIVERS
**********************************************************************
REL,%DVA76                 * TRAMCON segment #0 driver
REL,%DVA77                 * TRAMCON segment #1 driver
REL,%DVS72                 * 16-BIT RELAY OUTPUT DRIVER
REL,%DVR23                 * 7970 MAGNETIC TAPE UNIT
REL,%DVM33                 * 7912 disc , 65 Mbytes
REL,%DVT43                 * TOD/TBG CLOCK
REL,%4DP43                 * POWER FAIL
REL,%DVA66                 * 1000-1000 HDLC & 1000-3000 BISYNC
```

Figure B-1. (cont.)

```
REL,%MDV00                  * REMOTE I/O MAPPING DRIVER
REL,%DVX05                  * BACI CRT Driver
**********************************************************************
*                 MODULE
**********************************************************************
MAP OFF, MODULES
REL,%BMPG1                  * FILE MANAGER
REL,%BMPG2                  * D.RTR
REL,%WHZAT                  * SYSTEM STATUS PROGRAM
REL,%LGTAT                  * SYSTEM DISC LOG TABLE
REL,%$LDR                   * RELOCATING LOADER
REL,%ACCTS                  * ACCOUNT MAINTENANCE
REL,%AUTOR                  * Power Failure Recovery Untility
REL,%QUEUE                  * DS INTERRUPT REQUEST HANDLER
REL,%GRPM                   * DS General Req-Rep Processor
REL,%QCLM                   * DS ERROR MESSAGE LOGGER
REL,%RTRY                   * DS COMM. LINE RETRY PROCESSOR
REL,%RESSM                  * DS SSGA "res" mod for RTE-IVB
REL,%SMON1                  * SESSION MONITOR #1
REL,%SMON2                  * SESSION MONITOR #2
REL,%T5IDM                  * SHORT ID SEGMENT HANDLER
REL,%IOMAP                  * INTERFACE FOR MAPPED LUS
REL,%LUMAP                  * INTERFACE FOR MAPPED LUS
REL,%#SPLU                  * ENTRY POINT FOR REMOTE I/O MAPPING
REL,%DSMOD                  * DS NETWORK MODIFICATION
REL,%DINIS                  * NETWORK INITIALIZATION WITH SHUTDOWN
REL,%UPLIN                  * DS NETWORK WATCHDOG MONITOR
REL,%MATIC                  * DS Message Accounting
REL,%CSERR                  * CS80 Error Reporter
**********************************************************************
*                     LIBRARIES
**********************************************************************
REL,$MATH                   * System Mathematics Library
REL,$FOLDF                  * Fortran file I/O(FMGR file system)
REL,$FMP6,NOLIB
REL,$FLIB                   * Fortran system ind. library
REL,%BMPG3                  * FMP library
REL,$6SYLB                  * RTE-6VM System library
REL,$ACCLB                  * ACCOUNTS LIBRARY
REL,$DSMX6                  * DS
REL,$DSLB1                  * DS LIBRARY FOR ALL DS NODES
REL,$DSLB2                  * DS LIBRARY WHEN OTHER 1000'S IN NETWORK
REL,$DSLB3                  * DS LIBRARY IF NO LINKS TO 3000s
REL,$DSRR                   * DS RE-ROUTING LIBRARY
REL,$DSSM                   * DS LIBRARY WHEN HAVE SESSION MONITOR
REL,$FDSLB                  * FTN WITH DS LIB
REL,$DSMA                   * DS LIBRARY WHEN WANT MESSAGE ACCTG
REL,$IB6A                   * HPIB LIBRARY
```

Figure B-1. (cont.)

```
REL,%DBUGR                    * USER DEBUG ROUTINE
REL,$LDRLN                    * LOADER LIBRARY
REL,$MLSLB                    * MULTI LEVEL LOADER LIBRARY
REL,$UTLIB                    *
REL,%DECAR                    * DECIMAL STRING ARITHMETIC
/E
**********************************************************************
*       PROGRAM PARAMETERS
**********************************************************************
WHZAT,1,2                        * MEMORY RESIDENT-PRIORITY OF 2
IOMAP,19                         * CHANGE FROM RT DISC RES TO BG
LUMAP,19                         * CHANGE FROM RT DISC RES TO BG
FMGR,3,100                       * BG PRI 100
AUTOR,4,10                       * Background, NO Table Area 2, Priority 10
LGOFF,3,102                      * LGOFF PRIORITY BELOW FMGR
T5IDM,1                          * MEMORY RESIDENT
MATIC,17                         * MEMORY RESIDENT
PVM00,13                         * make type 13, so goes in Table Area II
/E                               * TERMINATE PARAMETER INPUT
**********************************************************************
*               ENTRY POINT CHANGES
**********************************************************************
*               FORTRAN77 COMPILER
Z$DBL,RP,4                       * 4 WORD DOUBLE PRECISION IS DEFAULT
Z$INT,RP,1                       * 16-bit INTEGERS ARE DEFAULT
Z$LPP,RP,73                      * 59 LINES PER PAGE DEFAULT
Z$F67,RP,7                       * COMPILER DEFAULTS TO 77 MODE
Z$CDS,RP,0                       * non-cds code generation
Z$CWD,RP,1                       * 6-bit system lu's in read and write
                                 * statements with control bits honored
**********************************************************************
*          EMA/VMA FIRMWARE EQUIVALENTS
**********************************************************************
.PMAP,RP,105240                  * MAP VMA/EMA PAGE IN MAP REG
$LOC,RP,105241                   * MEMORY-RESIDENT NODES LOAD ON CALL
.IMAP,RP,105250                  * SINGLE INT FTN4X ARRAY CALC + MAP
.IMAR,RP,105251                  * SINGLE INT SUBSCRIPT ARRAY CALC
.JMAP,RP,105252                  * DOUBLE INT FTN4X ARRAY CALC + MAP
.JMAR,RP,105253                  * DOUBLE INT SUBSCRIPT ARRAY CALC
.LPXR,RP,105254                  * TWO DEF POINTER ADD AND MAP
.LPX,RP,105255                   * A- AND B- REG. POINTER + DEF OFFSET
*                                * AND MAP
.LBPR,RP,105256                  * ONE DEF POINTER AND MAP
.LBP,RP,105257                   * MAP POINTER IN A- AND B-REGISTER
**********************************************************************
*          Operating System Firmware For E- and F-Series
**********************************************************************
$LIBR,RP,105340                  * EMULATE SYSTEM ENTRY $LIBR
$LIBX,RP,105341                  * EMULATE SYSTEM ENTRY $LIBX
```

Figure B-1. (cont.)

```
.FNW,RP,105345                 * FIND WORD WITH USER INCREMENT
.LLS,RP,105347                 * LINKED LIST SEARCH
.CPM,RP,105352                 * COMPARE WORDS IN MEMORY
.ENTN,RP,105354                * ENTRY POINT RESOLVER
.ENTC,RP,105356                * ENTRY POINT RESOLVER
*********************************************************************
*           SCIENTIFIC INSTRUCTION SET (SIS)
*********************************************************************
TAN,RP,105320
SQRT,RP,105321
ALOG,RP,105322
ATAN,RP,105323
COS,RP,105324
SIN,RP,105325
EXP,RP,105326
ALOGT,RP,105327
TANH,RP,105330
DPOLY,RP,105331
/CMRT,RP,105332
/ATLG,RP,105333
FPWR,RP,105334
.TPWR,RP,105335
*********************************************************************
*           FAST FORTRAN (FFP)
*********************************************************************
CLRIO,RP,2001
DBLE,RP,105201
SNGL,RP,105202
.DFER,RP,105205
.XPAK,RP,105206
.BLE,RP,105207
.NGL,RP,105214
.XCOM,RP,105215
..DCM,RP,105216
DDINT,RP,105217
.XFER,RP,105220
.GOTO,RP,105221
..MAP,RP,105222
.ENTR,RP,105223
.ENTP,RP,105224
.PWR2,RP,105225
.FLUN,RP,105226
$SETP,RP,105227
.PACK,RP,105230
.CFER,RP,105231
..FCM,RP,105232
..TCM,RP,105233
*********************************************************************
*          HFPP - TWO WORD
```

Figure B-1. (cont.)

```
***********************************************************************
.FIXD,RP,105104
.FLTD,RP,105124
***********************************************************************
*          HFPP - THREE WORD
***********************************************************************
.XADD,RP,105001
.XSUB,RP,105021
.XMPY,RP,105041
.XDIV,RP,105061
.XFXS,RP,105101
.DINT,RP,105101
.XFXD,RP,105105
.XFTS,RP,105121
.IDBL,RP,105121
.XFTD,RP,105125
***********************************************************************
*          HFPP FOUR WORD
***********************************************************************
.TADD,RP,105002
.TSUB,RP,105022
.TMPY,RP,105042
.TDIV,RP,105062
.TFXS,RP,105102
.TINT,RP,105102
.TFXD,RP,105106
.TFTS,RP,105122
.ITBL,RP,105122
.TFTD,RP,105126
***********************************************************************
*          DOUBLE WORD INTEGER
***********************************************************************
.DAD,RP,105014
.DSB,RP,105034
.DMP,RP,105054
.DDI,RP,105074
.DSBR,RP,105114
.DDIR,RP,105134
.DNG,RP,105203
.DIN,RP,105210
.DDE,RP,105211
.DIS,RP,105212
.DDS,RP,105213
.DCO,RP,105204
***********************************************************************
*          VECTOR INSTRUCTION SET FIRMWARE EQUIVALENTS
***********************************************************************
.VECT,RP,101460
VPIV,RP,101461
```

Figure B-1. (cont.)

```
VABS,RP,101462
VSUM,RP,101463
VNRM,RP,101464
VDOT,RP,101465
VMAX,RP,101466
VMAB,RP,101467
VMIN,RP,101470
VMIB,RP,101471
VMOV,RP,101472
VSWP,RP,101473
.DVCT,RP,105460
DVPIV,RP,105461
DVABS,RP,105462
DVSUM,RP,105463
DVNRM,RP,105464
DVDOT,RP,105465
DVMAX,RP,105466
DVMAB,RP,105467
DVMIN,RP,105470
DVMIB,RP,105471
DVMOV,RP,105472
DVSWP,RP,105473
/E                                * TERMINATE ENTRY POINT CHANGES
**********************************************************************
*                  Alias Name Section                                *
**********************************************************************
/E                                * Terminate alias name section
**********************************************************************
*           EQUIPMENT TABLE ENTRIES
**********************************************************************
* 10  fem
* 11  tbg
12,DVM33,D                        * EQT 1 - 7912 DISC (System)
13,DVX05,X=13                     * EQT 2 - SEGMENT CONSOLE #1
14,DVX05,X=13                     * EQT 3 - Segment Console #2
15,DVA76,X=13                     * EQT 4 - SEGMENT # 0
16,DVA77,X=13                     * EQT 5 - SEGMENT # 1
17,DVX05,X=13                     * EQT 6 - CRT #3
20,DVA66,X=12                     * EQT 7 - DS LINK #1 (TX)
20,DVA66                          * EQT 8 - DS LINK #1 (RX)
21,DVA66,X=12                     * EQT 9 - DS LINK #2 (TX)
21,DVA66                          * EQT 10 - DS LINK #2 (RX)
22,DVS72,T=3000                   * EQT 11 - RELAY OUTPUT (DUMMY)
23,DVX05,X=13                     * EQT 12 - Unused
25,DVX05,X=13                     * EQT 13 - Terminal interface (BACI)
71,DVV00                          * EQT 14 - REMOTE I/O RESERVED LU
71,DVV00,X=7                      * EQT 15 - MAPPING EQT #1
71,DVV00,X=7                      * EQT 16 - MAPPING EQT #2
71,DVV00,X=7                      * EQT 17 - MAPPING EQT #3
```

Figure B-1. (cont.)

```
11,DVT43,S                         * EQT 18 - TOD/TBG CLOCK
24,DVX05,X=13                      * EQT 19 - Unused
4,DVP43,M                          * EQT 20 - POWER FAIL
/E
***************************************************************************
*          DEVICE REFERENCE TABLE
***************************************************************************
2,0                                * LU 1 - SYSTEM CONSOLE
1,1                                * LU 2 - Disc, 7912, 15 Mbytes (system)
0                                  * LU 3 - unused
2,1                                * LU 4 - RIGHT CTU SYSTEM CONSOLE
2,2                                * LU 5 - LEFT CTU SYSTEM CONSOLE
2,4                                * LU 6 - PRINTER ON SYSTEM CONSOLE
0                                  * LU 7 - unused
1,0                                * LU 8 - disc subchannel 0 (CTD)
0                                  * LU 9 - unused
1,2                                * LU 10 - Disc,7912,50 Mbytes,(data files)
0                                  * LU 11 - unused
0                                  * LU 12 - unused
20                                 * LU 13 - POWER FAIL
11                                 * LU 14 - RELAY OUTPUT
4                                  * LU 15 - SEGMENT # 0
5                                  * LU 16 - SEGMENT # 1
7                                  * LU 17 - DS CHANNEL #1 (TX)
8                                  * LU 18 - DS CHANNEL #1 (RX)
9                                  * LU 19 - DS CHANNEL #2 (TX)
10                                 * LU 20 - DS CHANNEL #2 (RX)
0                                  * LU 21 - unused
0                                  * LU 22 - unused
0                                  * LU 23 - unused
0                                  * LU 24 - unused
2,0                                * LU 25 - TRAMCON CRT #1
3,0                                * LU 26 - TRAMCON CRT #2
6,0                                * LU 27 - TRAMCON CRT #3
13,0                               * LU 28 - MUX port
0                                  * LU 29 - unused
0                                  * LU 30 - unused
18                                 * LU 31 - TOD/TBG CLOCK
14                                 * LU 32 - REMOTE I/O RESERVED LU
15                                 * LU 33 - MAPPING LU 1
16                                 * LU 34 - MAPPING LU 2
17                                 * LU 35 - MAPPING LU 3
0                                  * LU 36 - unused
0                                  * LU 37 - Reserved for UP
0                                  * LU 38 - Reserved for UP
0                                  * LU 39 - Reserved for UP
0                                  * LU 40 - Reserved for UP
0                                  * LU 41 - Reserved for UP
2,4                                * LU 42 - CRT #1 Printer
```

Figure B-1. (cont.)

```
3,4                             * LU 43 - CRT #2 Printer
6,4                             * LU 44 - CRT #3 Printer
13,4                            * LU 45 - CRT #4 Printer
0                               * LU 46 - unused
0                               * LU 47 - unused
0                               * LU 48 - unused
0                               * LU 49 - unused
0                               * LU 50 - unused
0                               * LU 51 - unused
0                               * LU 52 - unused
0                               * LU 53 - unused
0                               * LU 54 - unused
0                               * LU 55 - unused
0                               * LU 56 - unused
0                               * LU 57 - unused
0                               * LU 58 - unused
0                               * LU 59 - unused
0                               * LU 60 - unused
/E                              * TERMINATE DRT
****************************************************************************
*                INTERRUPT TABLE
*************************************************************************
4,ENT,$POWR                     * POWER FAIL
*10,fem                         * fem for vma firmware
*11,tbg                         * time base generator
12,EQT,1                        * 7912 disc, 65 Mbytes
13,PRG,PRMPT                    * System Console
14,PRG,PRMPT                    * EXTRA TERMINAL #1
15,EQT,4                        * SEGMENT # 0
16,EQT,5                        * SEGMENT # 1
17,PRG,PRMPT                    * CRT #3
20,EQT,7                        * DS CHANNEL #1
21,EQT,9                        * DS CHANNEL #2
22,EQT,11                       * RELAY OUTPUT
23,PRG,PRMPT                    * Unused terminal port
24,PRG,PRMPT                    * Unused terminal port
25,PRG,PRMPT                    * Terminal Interface (BACI)
71,PRG,PRMPT
/E
***************************************************************************
*              SYSTEM BOUNDARIES
***************************************************************************
3                               * CHANGE DRIVER PART. SIZE? (YES)
0                               * CHANGE RT COMMON? (NO)
1                               * CHANGE BG COMMON? (1 EXTRA PAGE)
64                              * # OF I/O CLASSES
24                              * # OF LU MAPPINGS
40                              * # OF RESOURCE NUMBERS
100,400                         * BUFFER LIMITS
```

Figure B-1. (cont.)

```
40                                  * # OF BLANK ID SEGMENTS
25                                  * # OF BLANK SHORT ID SEGMENTS
25                                  * # OF BLANK ID EXTENSIONS
14                                  * MAXIMUM NUMBER OF PARTITIONS
************************************************************************
*               PARTITION DEFINITION
************************************************************************
71                                  * EXTRA SAM
32,RT,R                             * PARTITION 1 (FOR D.RTR)
5,BG                                * PARTITION 2
5,BG                                * PARTITION 3
12,BG                               * PARTITION 4
14,BG                               * PARTITION 5
27,BG                               * PARTITION 6
32,BG                               * PARTITION 7
32,BG                               * PARTITION 8
46,BG                               * PARTITION 9
NO                                  * NO subpartitions
46,BG                               * PARTITION 10
NO                                  * NO subpartitions
190,BG                              * PARTITION 11
NO                                  * NO subpartitions
/E
**********************************************************************
*       MODIFY PROGRAM PAGE REQUIREMENTS
**********************************************************************
FMGR,16
LOADR,27
ACCTS,18
D.RTR,32
/E
**********************************************************************
*        SHAREABLE EMA PARTITIONS
**********************************************************************
11,SHAR1
/E
*         SHAREABLE EMA PROGRAMS
/E
*         ASSIGN PROGRAM PARTITIONS
D.RTR,1                             * D.RTR ASSIGNED FIRST PARTITION
/E
```

Figure B-1. (cont.)

APPENDIX C: PROCEDURE FILE *LOAD6

```
:SV,1,9,IH
:* *LOAD6 - Load System Utilities, Software Development Tools and TRAMCON
:*          Segmented programs.  This must be done immediately after
:*          SWITCHING to a newly GENERATED system.
:* The programs loaded here are grouped into the following sections:
:*        System Manager Utilities
:*        System Utilities
:*        Program Development Utilities
:*        File System Utilities
:*        Help Utilities
:*        Backup Utilities
:*        Diagnostic and Disc Formatting Utilities
:*        Distributed System (DS) Programs
:*        TRAMCON Segmented Programs
:*
:* Except in cases where the program is permanently loaded, or LOADR is
:* required for some other reason, LINK is used to load all programs.
:* This is done for two reasons: 1) LINK is fast and creates one type 6
:* file that contains the program and all the segments; 2) for those
:* programs that do not have a load command file, LINK can accept multiple
:* files and commands in the run-string.
:*
:* For programs that are segmented, a list of segment names is
:* given.  If the LOADR is used, this command file will SP the main
:* and all the segments.
:*****************************************************************
:**                       N O T E                               *
:** Before any software can be loaded, the programs LINK and     *
:** LINDX must be loaded and the new revision of library $FMP6    *
:** must be indexed into the LINK SNAP file using LINDX.         *
:DP,Loading LINK
:RU,LOADR,#LINK
:SP,LINK
:SP,LINK1
:SP,LINK2
:SP,LINK3
:SP,LINK4
:SP,LINK5
:DP,Loading LINDX
:RU,LOADR,#LINDX
:SP,LINDX
:SP,LIND1
:SP,LIND2
:SP,LIND3
:** Now create the snap file for LINK
:** Ask if he wants to include any libraries other than $FMP6
```

Figure C-1. FMGR procedure file *LOAD6.

```
:DP,Do you want to include any libraries in the snap file
:DP,other than $FMP6?
:DP,(answer "TR,,YES" or "TR,,NO")
:PAUSE
:** upshift and blank-fill answer
:CA,-35:P,-35P,AND,157400B,OR,40B
:IF,-35P,EQ,54440B,      1
:IF,,EQ,,      9
:**ENDIF
:DP,Execute the following command appending the libraries you want,
:DP,then return to me with the "TR" command.
:DP,
:DP,RU,LINDX,SYSTEM,SNAP.6,$FMP6,<library>,<library>,...,+NL
:DP,
:PAUSE
:IF,,EQ,,      5
:**
:DP,Indexing the system and building SNAP.6 with $FMP6.
:RU,LINDX,SYSTEM,SNAP.6,$FMP6,+NL
:IF,,EQ,,      1
:**
:RU,LINDX,SYSTEM,SNAP.6,$FMP6,+NL
:*
:* System Manager Utilities
:* -----------------------
:* RT6GN  segments: RT6G1,RT6G2,RT6G3,RT6G4,RT6G5,RT6G6,RT6G7,RT6G8,RT6G9
:RU,LINK,#RT6GN,RT6GN::2
:*
:* SWTCH  segments: SWSG1,SWSG2,SWSG3
:RU,LINK,#SWTCH,SWTCH::2
:*
:* System Utilities
:* ---------------
:RU,LINK,%LUPRN,LUPRN::2
:*
:* Program Development Utilities
:* ----------------------------
:RU,LINK,%DRREL,$LDRLN,+SZ:+2,DRREL::2
:RU,LINK,%DRRPL,+SZ:+2,DRRPL::2
:*
:* EDIT   segments: EDIT0,EDIT1,EDIT2,EDIT3,EDIT4
:RU,LINK,#ED1K6,EDIT::2
:RU,LINK,%INDXR,INDXR::2
:*
:* MACRO  segments: MACR0,MACR1,MACR2,MACR3,MACR4,MACR5,MACR6,MACR7
:RU,LINK,#MACRO,MACRO::2
:*
```

Figure C-1. (cont.)

```
:* MLLDR  segments: MLLD1,MLLD2,MLLD3,MLLD4
:*        (MLLDR may not always load because of an overflow of base
:*         page links.  If this occurs (LOADR L-OV BSE error), follow
:*         the instructions in the load command file #MLLD6 to correct
:*         the situation.)
:RU,LOADR,#MLLD6
:SP,MLLDR::2
:SP,MLLD1::2
:SP,MLLD2::2
:SP,MLLD3::2
:SP,MLLD4::2
:*
:* SGMTR  segments: SGMT1,SGMT2,SGMT3,SGMT4,SGMT5,SGMT6
:RU,LOADR,#SGMTR
:SP,SGMTR::2
:SP,SGMT1::2
:SP,SGMT2::2
:SP,SGMT3::2
:SP,SGMT4::2
:SP,SGMT5::2
:SP,SGMT6::2
:*
:* SXREF  segments: SXRE1,SXRE2,SXRE3,SXRE4,SXRE5,SXRE6
:RU,LINK,#SXREF,SXREF::2
:*
:* File System Utilities
:* --------------------
:RU,LINK,#MERGE,MERGE::2
:RU,LINK,#OLDRE,OLDRE::2
:RU,LINK,#SCOM,SCOM::2
:*
:* Help Utilities
:* -------------
:RU,LINK,%CMD,$PLIBN,+LB,+SZ:28,CMD::2
:RU,LINK,%GENIX,$PLIBN,+EB,+SZ:30,GENIX::2
:RU,LINK,%HELP,$PLIBN,HELP::2
:*
:* Backup Utilities
:* ---------------
:* FC     segments: FC000,FC001,FC002,FC003,FC004,FC005,FC006
:RU,LINK,#FC6,FC::2
:RU,LINK,#TF,TF::2
:*
:* Diagnostic and Disc Formatting Utilities
:* ---------------------------------------
:RU,LINK,#FORMC,FORMC::2
:RU,LINK,#FORMT,FORMT::2
:RU,LINK,%TVVER,%TVLIB,TVVER::2
:*
```

Figure C-1. (cont.)

```
:* Distributed System (DS) Programs
:* --------------------------------
:TR,(DS,REMAN
:**
:** PROGRAM TO PROGRAM COMM SLAVE MONITOR
:TR,(DS,PTOPM
:**
:** REMOTE EXEC MONITOR
:TR,(DS,EXECM
:**
:** REMOTE 'SCHEDULE WITH WAIT ' MONITOR
:TR,(DS,EXECW
:**
:** REMOTE RTE OPERATOR COMMAND PROCESSOR
:TR,(DS,OPERM
:**
:** DS INFORMATION UTILITY
:TR,(DS,DSINF
:**
:** REMOTE DIRECTORY LISTER
:TR,(DS,DLIS1
:**
:** REMOTE FILE ACCESS MONITOR
:TR,(DS,RFAM2
:**
:** DYNAMIC MESSAGE REROUTING
:TR,(DS,#SEND
:**
:** REMOTE SESSION MONITOR
:TR,(DS,RSM
:**
:** REMOTE I/O MAPPING MODULES
:TR,(DS,SYSAT
:TR,(DS,LUQUE
:**
:** VIRTUAL CONTROL PANEL MONITOR
:TR,(DS,VCPMN
:**
:** SLAVE MONITOR FOR REMOTE DOWN-LOAD FROM CBL
:TR,(DS,PROGL
:**
:** DS VERSION OF EDITR
:TR,(DS,EDI6D
:*
:**********************
:** Load Program CF   *
:OF,CF
:RU,MLLDR,#CF
:PU,CF
```

Figure C-1. (cont.)

```
:SP,CF
:PU,@CF
:**********************
:** Load Program DT   *
:OF,DT
:RU,MLLDR,#DT
:PU,DT
:SP,DT
:PU,@DT
:**********************
:** Load Program INIT *
:RU,MLLDR,#INIT
:PU,INIT
:SP,INIT
:PU,@INIT
:**********************
:** Load Program MTRP *
:OF,MTRP
:RU,MLLDR,#MTRP
:PU,MTRP
:SP,MTRP
:PU,@MTRP
:**********************
:** Load Program PLRP *
:OF,PLRP
:RU,MLLDR,#PLRP
:PU,PLRP
:SP,PLRP
:PU,@PLRP
:**********************
:** Load Program SR   *
:OF,SR
:RU,MLLDR,#SR
:PU,SR
:SP,SR
:PU,@SR
:*
:SV,9G,,IH
```

Figure C-1. (cont.)

```
#RT6GN - LINK command file for RT6GN,  92084-17268  rev.2440
                EB
                EC
                EM,100
                LI,$R6GNL
                * LI,$FMP6::A85
                * LI,$FLIB::A85
                RE,%RT6GN
                end rt6gn

#SWTCH - LINK command file for SWTCH,  92084-17039  REV.2440
                ps
                LI,$DTCLB
                LI,$DSCLB
                * LI,$FMP6          ,,, FOR OLDER SYSTEMS
                RE,%SSTCH
                end swtch

#ED1K6 - LINK command file for EDIT, 92074-17003 REV.2440
                OP,EB
                OP,PE
                SZ,32, * 24 TO 32 PAGES. LARGER = FASTER.
                LIB,$ED1K6
                RE,%EDITA
                RE,%EDITB
                EN

#MACRO - LINK command file for MACRO,  92059-17004 REV.2340

        NOTE: Sizing macro to 32 pages will make it run fastest.
              21 pages is minimum.
                EB
                SZ,32
                RE,%MACRO
                RE,%MACR0
                RE,%MACR3
                RE,%MACR7
                RE,%MACR1
                RE,%MACR2
                RE,%MACR4
                RE,%MACR5
                RE,%MACR6
                EN
```

Figure C-2. LINK and MLLDR command files used by *LOAD6.

```
#MLLD6 - LOADR command file for MLLDR,  92084-17189 REV.2226
                    EBCP
                    SZ,32
                    LI,$RBLIB
                    RE,%MLLDR
                    *LO,10000B ,USE IF LOADR GIVES L-OV BSE
                    RE,%M.LIB
                    *LO,16000B ,USE IF LOADR GIVES L-OV BSE
                    RE,%MLLDA
                    *LO,22000B ,USE LOADR GIVES L-OV BSE
                    RE,%MLLDB
                    *LO,40000B ,USE IF LOADR GIVES L-OV BSE
                    EN

#SGMTR - LOADR command file for SGMTR,  92084-17106 REV.2121
                    WS,5
                    LI,$RBLIB
                    CPLBVM
                    RE,%SGMTR
                    EN

#MERGE - LINK command file for MERGE,  92084-17208 REV.2340
                    LI,$FMP6
                    RE,%MERGE
                    EN

#OLDRE - LINK command file for OLDRE,  92059-17002 REV.2213
                    RE,%OLDRE
                    RE,%ATRAN
                    EN

#SCOM - LINK command file for SCOM,   92084-17036 REV.2340
                    EB
                    LI,$FMP6
                    RE,%SCOM
                    EN

#FC6 - LINK command file for FC,  92084-17151 REV.2302
                    OP,EB
                    SZ,32
                    OP,MP
                    LI,$FCL1
                    LI,$FCL2
                    LI,$PLIB
                    RE,%FCM6
                    RE,%FC0
                    RE,%FC1
                    RE,%FC2
                    RE,%FC3
```

Figure C-2. (cont.)

```
                    RE,%FC4
                    RE,%FC5
                    RE,%FC6
                    EN
```

#TF - LINK command file for TF, 92077-17102 REV.2440

NOTE: The value in the EM command may be increased to allow larger copy command groups.

```
                    EB
                    EM,11
                    LI,$TFLIB
                    RE,%TF
                    EN
```

#FORMC - LINK command file for FORMC,   92077-17034 REV.2440

```
                    eb
                    li,$dtclb
                    li,$dsclb
                    li,$fmp6
                    re,%formc
                    re,%fc000
                    en
```

#FORMT - LINK command file for FORMT, 92084-17029 REV.2340

```
                    SZ,18
                    RE,%FORMT
                    SEA,$DSCLB
                    EN
```

#DS - Referenced by file (DS to load each DS program

```
                    BG
                    SH,SHAR1
                    LI,$PLIB
                    EN
```

Figure C-2. (cont.)

```
(DS - used to load each DS program,  written by ITS
                        :CA,2,1G,/,400B
                        :CA,3,1G
                        :CA,-27:P,-27P,AND,177B
                        :CA,-26:P,-26P,AND,177B
                        :CA,-25:P,0
                        :CA,3,3G,*,400B
                        :CA,-30:P,-30P,OR,-27P
                        :CA,-29:P,-29P,OR,-26P
                        :CA,-31:P,-31P,OR,22400B
                        :OF,1G
                        :RU,LOADR:IH,#DS,2G
                        :PU,10G
                        :SP,10G
                        :TR
```

Figure C-3. Procedure files referenced by *LOAD6.

APPENDIX D: PROGRAM "CHGREC" - USED TO CHANGE DISC FILE RECORD SIZE

```
$PASCAL  'Change disc file record size' , HEAPPARMS OFF $
$RECURSIVE OFF , RANGE OFF$
PROGRAM chgrec;                        {<880818.1550>}

CONST scode = 21586 {TR};  eof = -1;  temp_name = 'XXXXXX';
      newreclen =

{ENTER length (in words) of NEW record}
 670;
{end ENTER}

$INCLUDE '[RECR3'$

size_array = ARRAY[1..2] OF INT;

new_record =

{ENTER NEW record definition here}
 statz_record;
{end ENTER}

old_record =

{ENTER the OLD record definition here}
 RECORD
 cnt_cmds: ARRAY[un..us , master_crt_ordinal] OF INT; {count cmds }
 transmission: ARRAY[master_segment_ordinal] OF
                 ARRAY[segment_remote_ordinal,msg_status] OF INT;
 END;
{end ENTER}

VAR i,j,k,crn,sectors,offset,nxtblk,nxtrec,ftype,reclen,err,len,recnbr: INT;

    size: size_array;
    new_rec: new_record;
    old_rec: old_record;
    $LINESIZE 500$ outunit: TEXT;  key: two_chars;
    olddcb,newdcb: data_control_block;
    old_name: six_chars;

{ENTER special purpose variables here}
    cmd: cmds; msg_st: msg_status;
{end ENTER}

PROCEDURE read_key $ALIAS'EXEC'$(f,lu: INT; buf:two_chars; l:INT);EXTERNAL;
```

Figure D-1. Source listing &CHREC.

```
PROCEDURE creat (dcb:data_control_block; VAR err: INT; nam: six_chars;
                 size: size_array; ftype,scode,crn: INT); EXTERNAL;

PROCEDURE locf (dcb:data_control_block;  VAR err,nxtrec,nxtblk: INT;
                VAR offset,sectors,lu,ftype,reclen: INT); EXTERNAL;

PROCEDURE openf (dcb:data_control_block;
                 VAR err: INT; nam: six_chars); EXTERNAL;

PROCEDURE closf $ALIAS'CLOSE'$ (dcb:data_control_block;
                 VAR err: INT; trunate: INT); EXTERNAL;

PROCEDURE readf (dcb:data_control_block; VAR err: INT;
                 VAR buf: old_record; reclen: INT; VAR len: INT); EXTERNAL;

PROCEDURE writf (dcb:data_control_block; VAR err: INT;
                 VAR buf: new_record; len: INT); EXTERNAL;

PROCEDURE namf (dcb:data_control_block; VAR err: INT;
                oldnam,newnam: six_chars; scode,crn: INT); EXTERNAL;

PROCEDURE purge (dcb:data_control_block; VAR err: INT;
                 nam: six_chars; scode,crn: INT); EXTERNAL;

BEGIN {chgrec}
old_name :=

{ENTER name of OLD file here}
 '(STATZ';
{end ENTER}

rewrite(outunit,'1',nocctl_shared);
openf(olddcb,err,old_name);
writeln(outunit,lf,lf,'NEWREC  Opening File ',old_name:6,
               '         err = ',err:6,lf,lf);
locf(olddcb,err,nxtrec,nxtblk,offset,sectors,crn,ftype,reclen);
IF ftype > 2 THEN BEGIN size[1] := -1; size[2] := 128 END
ELSE
  BEGIN size[1] := (sectors+1) DIV 2;
  size[1] := size[1] +

{ENTER more or less blocks to overall file size due to record size change}
 1;
{end ENTER}

  size[2] := newreclen
  END;
```

Figure D-1. (cont.)

```
prompt(outunit,'Changing record size for File ', old_name:6,lf,lf,ret,
               'Blocks    = ', size[1]:6,lf,ret,
               'oldreclen = ', reclen:6,lf,ret,
               'newreclen = ', size[2]:6,lf,ret,
               'crn       = ', crn:6,lf,ret,
               'ftype     = ', ftype:6,lf,ret,
               'err       = ', err:6,lf,ret,lf,
               'OK to Proceed ?(Y or N)',bell);
 read_key(1,65,key,-1);
 IF (key[1] = 'Y') or (key[1] = 'y') THEN
   BEGIN creat(newdcb,err,temp_name,size,ftype,scode,crn); len := 0;
   recnbr := 1;  readf(olddcb,err,old_rec,reclen,len);
   prompt(outunit,ret,'Reading Record Number ', recnbr:6);
   WHILE len <> eof DO
     BEGIN
{ENTER code to transfer old_rec to new_rec }
     FOR cmd := un TO us DO
       FOR i := 0 TO max_crts_per_master-1 DO
         new_rec.cnt_cmds[cmd,i] := 0;
     FOR i := 0 TO max_segments_per_master-1 DO
       FOR j := 0 TO max_remotes_per_segment-1 DO
         FOR msg_st := polls TO no_ans DO
           new_rec.transmission[i][j,msg_st] :=
             old_rec.transmission[i][j,msg_st];
{end ENTER}
     writf(newdcb,err,new_rec,newreclen);
     readf(olddcb,err,old_rec,reclen,len);
     IF len <> eof THEN
       BEGIN recnbr := recnbr+1; prompt(outunit,esc,'&a-6C',recnbr:6) END
     END;
   purge(olddcb,err,old_name,scode,crn); {Purge OLD File}
   writeln(outunit,lf,ret,lf,'Purging File ',old_name:6,'  err =',err:6);
   write(outunit,lf,lf,'Closing the NEW ',old_name:6,' File');
   IF ftype > 2 THEN
     BEGIN locf(newdcb,err,nxtrec,nxtblk,offset,sectors,crn,ftype,reclen);
     j := (sectors DIV 2); k := nxtblk+1; i := j - k - 1;
     write(outunit,' , Truncating from ',j:6,' to ',k:6,' Blocks');
     closf(newdcb,err,i)
     END;
   namf(newdcb,err,temp_name,old_name,scode,crn);
   writeln(outunit,ret,lf,lf,lf,'Renaming File ',temp_name:6,
                ' to ',old_name:6,'   err =',err:6)
   END;
writeln(outunit,lf,lf,ret,'chgrec end',bell)
{COMMENT brackets should be removed from the following line for the first
 compilation so that the NEW record size (in words) can be determined from
 the compiler TABLES in the listing file 'CHREC.  To generate this listing
 file, the compiler should be run as follows: RU,P,&CHREC,-,%CHREC::10  }
{$LIST ON, TABLES ON$}
END.  {chgrec}
```

Figure D-1. (cont.)

APPENDIX E: DATALOK10, MODELS 1D AND 1E, RAW RESPONSE FORMATS

This appendix presents the raw response formats for the DATALOK10, models 1D and 1E, remote units. The responses are listed in the order they are received, with the byte number listed on the left. For example, byte number 5 is the fifth byte in the response. The 8 bits in each byte are listed just to the right of the byte number. The 8 bits are shown in reverse order. That is, bit 0 is received first and bit 7 is received last. Bit 7 is the Parity bit. The DATALOK10 remote unit uses EVEN parity.

Each alarm/status byte contains 6 bits of data. The CATEGORY and TWO-STATE assignments for these 6 bits are detailed just below the given byte.

Datalok 10, model 1E response format

	b i t s	
byte	76543210	Description
1	P100xxxx	Polling ID - byte 1 (High-order 4 bits of Polling ID)
2	P100yyyy	Polling ID - byte 2 (Low-order 4 bits of Polling ID)
3	11010001	Polling ID - byte 3 (Always set to ASCII "A")
4	P0xxxxxx	1J4 - byte 1 of 1st 18-point encoder
	1	bit 0 - category 0, two-state 52
	2	1 - category 0, two-state 53
	3	2 - category 0, two-state 54
	4	3 - category 0, two-state 55
	5	4 - category 0, two-state 56
	6	5 - category 0, two-state 57
5	P0xxxxxx	1J4 - byte 2 of 1st 18-point encoder
	7	bit 0 - category 0, two-state 58
	8	1 - category 0, two-state 59
	9	2 - category 0, two-state 60
	10	3 - category 0, two-state 61
	11	4 - category 0, two-state 62
	12	5 - category 0, two-state 63
6	P0xxxxxx	1J4 - byte 3 of 1st 18-point encoder
	13	bit 0 - category 0, two-state 64
	14	1 - category 0, two-state 65
	15	2 - category 0, two-state 66
	16	3 - category 0, two-state 67
	17	4 - category 0, two-state 68
	18	5 - category 0, two-state 69
7	P0xxxxxx	1J5 - byte 1 of 2nd 18-point encoder
	19	bit 0 - category 1, two-state 52
	20	1 - category 1, two-state 53
	21	2 - category 1, two-state 54
	22	3 - category 1, two-state 55
	23	4 - category 1, two-state 56
	24	5 - category 1, two-state 57
8	P0xxxxxx	1J5 - byte 2 of 2nd 18-point encoder
	25	bit 0 - category 1, two-state 58
	26	1 - category 1, two-state 59
	27	2 - category 1, two-state 60
	28	3 - category 1, two-state 61

```
    29            4 - category 1, two-state 62
    30            5 - category 1, two-state 63
 9  P0xxxxxx  1J5 - byte 3 of 2nd 18-point encoder
    31        bit 0 - category 1, two-state 64
    32            1 - category 1, two-state 65
    33            2 - category 1, two-state 66
    34            3 - category 1, two-state 67
    35            4 - category 1, two-state 68
    36            5 - category 1, two-state 69
10  P0xxxxxx  1J6 - byte 1 of 3rd 18-point encoder
    37        bit 0 - category 2, two-state 52
    38            1 - category 2, two-state 53
    39            2 - category 2, two-state 54
    40            3 - category 2, two-state 55
    41            4 - category 2, two-state 56
    42            5 - category 2, two-state 57
11  P0xxxxxx  1J6 - byte 2 of 3rd 18-point encoder
    43        bit 0 - category 2, two-state 58
    44            1 - category 2, two-state 59
    45            2 - category 2, two-state 60
    46            3 - category 2, two-state 61
    47            4 - category 2, two-state 62
    48            5 - category 2, two-state 63
12  P0xxxxxx  1J6 - byte 3 of 3rd 18-point encoder
    49        bit 0 - category 2, two-state 64
    50            1 - category 2, two-state 65
    51            2 - category 2, two-state 66
    52            3 - category 2, two-state 67
    53            4 - category 2, two-state 68
    54            5 - category 2, two-state 69
13  P0xxxxxx  1J7 - byte 1 of 4th 18-point encoder
    55        bit 0 - site category, two-state 12
    56            1 - site category, two-state 13
    57            2 - site category, two-state 14
    58            3 - site category, two-state 15
    59            4 - site category, 16
    60            5 - site category, 17
14  P0xxxxxx  1J7 - byte 2 of 4th 18-point encoder
    61        bit 0 - site category, 18
    62            1 - site category, 19
    63            2 - site category, 20
    64            3 - site category, 21
    65            4 - site category, 22
    66            5 - site category, 23
15  P0xxxxxx  1J7 - byte 3 of 4th 18-point encoder
    67        bit 0 - site category, 24
    68            1 - site category, 25
    69            2 - site category, 26
    70            3 - site category, 27
    71            4 - site category, 28
    72            5 - site category, 29
16  P0xxxxxx  1J8 - byte 1 of 1st 12-point encoder
```

73		bit 0 - site category, two-state 0
74		1 - site category, two-state 1
75		2 - site category, two-state 2
76		3 - site category, two-state 3
77		4 - site category, two-state 4
78		5 - site category, two-state 5
17	POxxxxxx	1J8 - byte 2 of 1st 12-point encoder
79		bit 0 - site category, two-state 6
80		1 - site category, two-state 7
81		2 - site category, two-state 8
82		3 - site category, two-state 9
83		4 - site category, two-state 10
84		5 - site category, two-state 11
18	POxxxxxx	1J9 - byte 1 of 2nd 12-point encoder
85		bit 0 - category 0, two-state 0
86		1 - category 0, two-state 1
87		2 - category 0, two-state 2
88		3 - category 0, two-state 3
89		4 - category 0, two-state 4
90		5 - category 0, two-state 5
19	POxxxxxx	1J9 - byte 2 of 2nd 12-point encoder
91		bit 0 - category 0, two-state 6
92		1 - category 0, two-state 7
93		2 - category 0, two-state 8
94		3 - category 0, two-state 9
95		4 - category 0, two-state 10
96		5 - category 0, two-state 11
20	POxxxxxx	1J10 - byte 1 of 3rd 12-point encoder
97		bit 0 - category 0, two-state 12
98		1 - category 0, two-state 13
99		2 - category 0, two-state 14
100		3 - category 0, two-state 15
101		4 - category 0, two-state 16
102		5 - category 0, two-state 17
21	POxxxxxx	1J10 - byte 2 of 3rd 12-point encoder
103		bit 0 - category 0, two-state 18
104		1 - category 0, two-state 19
105		2 - category 0, two-state 20
106		3 - category 0, two-state 21
107		4 - category 0, two-state 22
108		5 - category 0, two-state 23
22	POxxxxxx	1J11 - byte 1 of 4th 12-point encoder
109		bit 0 - category 0, two-state 24
110		1 - category 0, two-state 25
111		2 - category 0, two-state 26
112		3 - category 0, two-state 27
113		4 - category 0, two-state 28
114		5 - category 0, two-state 29
23	POxxxxxx	1J11 - byte 2 of 4th 12-point encoder
115		bit 0 - category 0, two-state 30
116		1 - category 0, two-state 31
117		2 - category 0, two-state 32

```
  118           3 - category 0, two-state 33
  119           4 - category 0, two-state 34
  120           5 - category 0, two-state 35
24  P0xxxxxx  1J12 - byte 1 of 5th 12-point encoder
  121         bit 0 - category 1, two-state 0
  122           1 - category 1, two-state 1
  123           2 - category 1, two-state 2
  124           3 - category 1, two-state 3
  125           4 - category 1, two-state 4
  126           5 - category 1, two-state 5
25  P0xxxxxx  1J12 - byte 2 of 5th 12-point encoder
  127         bit 0 - category 1, two-state 6
  128           1 - category 1, two-state 7
  129           2 - category 1, two-state 8
  130           3 - category 1, two-state 9
  131           4 - category 1, two-state 10
  132           5 - category 1, two-state 11
26  P0xxxxxx  1J13 - byte 1 of 6th 12-point encoder
  133         bit 0 - category 1, two-state 12
  134           1 - category 1, two-state 13
  135           2 - category 1, two-state 14
  136           3 - category 1, two-state 15
  137           4 - category 1, two-state 16
  138           5 - category 1, two-state 17
27  P0xxxxxx  1J13 - byte 2 of 6th 12-point encoder
  139         bit 0 - category 1, two-state 18
  140           1 - category 1, two-state 19
  141           2 - category 1, two-state 20
  142           3 - category 1, two-state 21
  143           4 - category 1, two-state 22
  144           5 - category 1, two-state 23
28  P0xxxxxx  1J14 - byte 1 of 7th 12-point encoder
  145         bit 0 - category 1, two-state 24
  146           1 - category 1, two-state 25
  147           2 - category 1, two-state 26
  148           3 - category 1, two-state 27
  149           4 - category 1, two-state 28
  150           5 - category 1, two-state 29
29  P0xxxxxx  1J14 - byte 2 of 7th 12-point encoder
  151         bit 0 - category 1, two-state 30
  152           1 - category 1, two-state 31
  153           2 - category 1, two-state 32
  154           3 - category 1, two-state 33
  155           4 - category 1, two-state 34
  156           5 - category 1, two-state 35
30  P0xxxxxx  1J15 - byte 1 of 8th 12-point encoder
  157         bit 0 - category 2, two-state 0
  158           1 - category 2, two-state 1
  159           2 - category 2, two-state 2
  160           3 - category 2, two-state 3
  161           4 - category 2, two-state 4
  162           5 - category 2, two-state 5
```

```
31  POxxxxxx  1J15 - byte 2 of 8th 12-point encoder
  163         bit 0 - category 2, two-state 6
  164             1 - category 2, two-state 7
  165             2 - category 2, two-state 8
  166             3 - category 2, two-state 9
  167             4 - category 2, two-state 10
  168             5 - category 2, two-state 11
32  POxxxxxx  1J16 - byte 1 of 9th 12-point encoder
  169         bit 0 - category 2, two-state 12
  170             1 - category 2, two-state 13
  171             2 - category 2, two-state 14
  172             3 - category 2, two-state 15
  173             4 - category 2, two-state 16
  174             5 - category 2, two-state 17
33  POxxxxxx  1J16 - byte 2 of 9th 12-point encoder
  175         bit 0 - category 2, two-state 18
  176             1 - category 2, two-state 19
  177             2 - category 2, two-state 20
  178             3 - category 2, two-state 21
  179             4 - category 2, two-state 22
  180             5 - category 2, two-state 23
34  POxxxxxx  1J17 - byte 1 of 10th 12-point encoder
  181         bit 0 - category 2, two-state 24
  182             1 - category 2, two-state 25
  183             2 - category 2, two-state 26
  184             3 - category 2, two-state 27
  185             4 - category 2, two-state 28
  186             5 - category 2, two-state 29
35  POxxxxxx  1J17 - byte 2 of 10th 12-point encoder
  187         bit 0 - category 1, two-state 30
  188             1 - category 1, two-state 31
  189             2 - category 1, two-state 32
  190             3 - category 1, two-state 33
  191             4 - category 1, two-state 34
  192             5 - category 1, two-state 35
36  POxxxxxx  1J18 - byte 1 of 11th 12-point encoder
  193         bit 0 - category 0, two-state 36
  194             1 - category 0, two-state 37
  195             2 - category 0, two-state 38
  196             3 - category 0, two-state 39
  197             4 - category 0, two-state 40
  198             5 - category 0, two-state 41
37  POxxxxxx  1J18 - byte 2 of 11th 12-point encoder
  199         bit 0 - category 0, two-state 42
  200             1 - category 0, two-state 43
  201             2 - category 0, two-state 44
  202             3 - category 0, two-state 45
  203             4 - category 0, two-state 46
  204             5 - category 0, two-state 47
38  POxxxxxx  2J2 - byte 1 of 12th 12-point encoder
  205         bit 0 - category 0, two-state 48
  206             1 - category 0, two-state 49
```

207		2 - category 0, two-state 50
208		3 - category 0, two-state 51
209		4 - category 1, two-state 36
210		5 - category 1, two-state 37
39	P0xxxxxx	2J2 - byte 2 of 12th 12-point encoder
211		bit 0 - category 1, two-state 38
212		1 - category 1, two-state 39
213		2 - category 1, two-state 40
214		3 - category 1, two-state 41
215		4 - category 1, two-state 42
216		5 - category 1, two-state 43
40	P0xxxxxx	2J3 - byte 1 of 13th 12-point encoder
217		bit 0 - category 1, two-state 44
218		1 - category 1, two-state 45
219		2 - category 1, two-state 46
220		3 - category 1, two-state 47
221		4 - category 1, two-state 48
222		5 - category 1, two-state 49
41	P0xxxxxx	2J3 - byte 2 of 13th 12-point encoder
223		bit 0 - category 1, two-state 50
224		1 - category 1, two-state 51
225		2 - category 2, two-state 36
226		3 - category 2, two-state 37
227		4 - category 2, two-state 38
228		5 - category 2, two-state 39
42	P0xxxxxx	2J4 - byte 1 of 14th 12-point encoder
229		bit 0 - category 2, two-state 40
230		1 - category 2, two-state 41
231		2 - category 2, two-state 42
232		3 - category 2, two-state 43
233		4 - category 2, two-state 44
234		5 - category 2, two-state 45
43	P0xxxxxx	2J4 - byte 2 of 14th 12-point encoder
235		bit 0 - category 2, two-state 46
236		1 - category 2, two-state 47
237		2 - category 2, two-state 48
238		3 - category 2, two-state 49
239		4 - category 2, two-state 50
240		5 - category 2, two-state 51

Analog-to-Digital Parameters (16 channel A/D Mux card)

44	10000001	2J5, pin 1 - 1st Analog Value, **Parameter 0, category 0**
45	00000000	Scale ID (set to 0)
46	P00THHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
47	P000TTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
48	P000UUUU	Overflow bit(bit 4) and Units digit (BCD)
49	10000010	2J5, pin 2 - 2nd Analog Value, **Parameter 1, category 0**
50	00000000	Scale ID (set to 0)
51	P00THHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
52	P000TTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
53	P000UUUU	Overflow bit(bit 4) and Units digit (BCD)

54	00000011	2J5, pin 3 - 3rd Analog Value, **Parameter 0, category 1**
55	00000000	Scale ID (set to 0)
56	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
57	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
58	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
59	10000100	2J5, pin 4 - 4th Analog Value, **Parameter 1, category 1**
60	00000000	Scale ID (set to 0)
61	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
62	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
63	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
64	00000101	2J5, pin 5 - 5th Analog Value, **Parameter 0, category 2**
65	00000000	Scale ID (set to 0)
66	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
67	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
68	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
69	00000110	2J5, pin 6 - 6th Analog Value, **Parameter 1, category 2**
70	00000000	Scale ID (set to 0)
71	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
72	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
73	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
74	10000111	2J5, pin 7 - 7th Analog Value, **Parameter 0, Site Cat**
75	00000000	Scale ID (set to 0)
76	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
77	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
78	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
79	10001000	2J5, pin 8 - 8th Analog Value, **UNUSED**
80	00000000	Scale ID (set to 0)
81	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
82	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
83	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
84	00001001	2J5, pin 9 - 9th Analog Value, **Parameter 2, category 0**
85	00000000	Scale ID (set to 0)
86	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
87	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
88	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
89	00001010	2J5, pin 10 - 10th Analog Value, **Parameter 3, category 0**
90	00000000	Scale ID (set to 0)
91	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
92	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
93	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
94	10001011	2J5, pin 11 - 11th Analog Value, **Parameter 2, category 1**
95	00000000	Scale ID (set to 0)
96	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
97	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
98	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
99	00001100	2J5, pin 12 - 12th Analog Value, **Parameter 3, category 1**
100	00000000	Scale ID (set to 0)
101	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
102	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
103	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
104	10001101	2J5, pin 13 - 13th Analog Value, **Parameter 2, category 2**
105	00000000	Scale ID (set to 0)

106	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
107	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
108	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
109	10001110	2J5, pin 14 - 14th Analog Value, **Parameter 3, category 2**
110	00000000	Scale ID (set to 0)
111	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
112	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
113	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)

Digital Parameters (FEC Cards, 4 bytes per card)

114	P0xxxxxx	2J13,	Parameter 0, Category 0, low-order 6 bits
115	P0xxxxxx		Parameter 0, Category 0, high-order 6 bits
116	P0xxxxxx		Parameter 1, Category 0, low-order 6 bits
117	P0xxxxxx		Parameter 1, Category 0, high-order 6 bits
118	P0xxxxxx	2J14,	Parameter 2, Category 0, low-order 6 bits
119	P0xxxxxx		Parameter 2, Category 0, high-order 6 bits
120	P0xxxxxx		Parameter 3, Category 0, low-order 6 bits
121	P0xxxxxx		Parameter 3, Category 0, high-order 6 bits
122	P0xxxxxx	2J15,	Parameter 4, Category 0, low-order 6 bits
123	P0xxxxxx		Parameter 4, Category 0, high-order 6 bits
124	P0xxxxxx		Parameter 5, Category 0, low-order 6 bits
125	P0xxxxxx		Parameter 5, Category 0, high-order 6 bits
126	P0xxxxxx	2J16,	Parameter 6, Category 0, low-order 6 bits
127	P0xxxxxx		Parameter 6, Category 0, high-order 6 bits
128	P0xxxxxx		Parameter 7, Category 0, low-order 6 bits
129	P0xxxxxx		Parameter 7, Category 0, high-order 6 bits
130	P0xxxxxx	2J17,	Parameter 0, Category 1, low-order 6 bits
131	P0xxxxxx		Parameter 0, Category 1, high-order 6 bits
132	P0xxxxxx		Parameter 1, Category 1, low-order 6 bits
133	P0xxxxxx		Parameter 1, Category 1, high-order 6 bits
134	P0xxxxxx	2J18,	Parameter 2, Category 1, low-order 6 bits
135	P0xxxxxx		Parameter 2, Category 1, high-order 6 bits
136	P0xxxxxx		Parameter 3, Category 1, low-order 6 bits
137	P0xxxxxx		Parameter 3, Category 1, high-order 6 bits
138	P0xxxxxx	3J3,	Parameter 4, Category 1, low-order 6 bits
139	P0xxxxxx		Parameter 4, Category 1, high-order 6 bits
140	P0xxxxxx		Parameter 5, Category 1, low-order 6 bits
141	P0xxxxxx		Parameter 5, Category 1, high-order 6 bits
142	P0xxxxxx	3J4,	Parameter 6, Category 1, low-order 6 bits
143	P0xxxxxx		Parameter 6, Category 1, high-order 6 bits
144	P0xxxxxx		Parameter 7, Category 1, low-order 6 bits
145	P0xxxxxx		Parameter 7, Category 1, high-order 6 bits
146	P0xxxxxx	3J5,	Parameter 0, Category 2, low-order 6 bits
147	P0xxxxxx		Parameter 0, Category 2, high-order 6 bits
148	P0xxxxxx		Parameter 1, Category 2, low-order 6 bits
149	P0xxxxxx		Parameter 1, Category 2, high-order 6 bits
150	P0xxxxxx	3J6,	Parameter 2, Category 2, low-order 6 bits
151	P0xxxxxx		Parameter 2, Category 2, high-order 6 bits
152	P0xxxxxx		Parameter 3, Category 2, low-order 6 bits
153	P0xxxxxx		Parameter 3, Category 2, high-order 6 bits
154	P0xxxxxx	3J7,	Parameter 4, Category 2, low-order 6 bits

```
155  P0xxxxxx       Parameter 4, Category 2, high-order 6 bits
156  P0xxxxxx       Parameter 5, Category 2, low-order 6 bits
157  P0xxxxxx       Parameter 5, Category 2, high-order 6 bits
158  P0xxxxxx  3J8, Parameter 6, Category 2, low-order 6 bits
159  P0xxxxxx       Parameter 6, Category 2, high-order 6 bits
160  P0xxxxxx       Parameter 7, Category 2, low-order 6 bits
161  P0xxxxxx       Parameter 7, Category 2, high-order 6 bits

162  11111111  End of Transmission Character ( ASCII <DELETE> )
```

Datalok 10, model 1D Response format

	b i t s	
byte	76543210	Description
1	P100xxxx	Polling ID - byte 1 (High order 4 bits of Polling ID)
2	P100yyyy	Polling ID - byte 2 (Low order 4 bits of Polling ID)
3	11010001	Polling ID - byte 3 (Always set to ASCII "A")
4	P0xxxxxx	1J4 (strapped for 3 bytes) - byte 1 of 1st 18-point encoder
	1	bit 0 - site category, two-state unassigned
	2	1 - site category, two-state unassigned
	3	2 - site category, two-state unassigned
	4	3 - site category, two-state unassigned
	5	4 - site category, two-state unassigned
	6	5 - site category, two-state unassigned
5	P0xxxxxx	1J4 - byte 2 of 1st 18-point encoder
	7	bit 0 - category 0, two-state 56
	8	1 - category 0, two-state 57
	9	2 - category 1, two-state 56
	10	3 - category 1, two-state 57
	11	4 - category 2, two-state 56
	12	5 - category 2, two-state 57
6	P0xxxxxx	1J4 - byte 3 of 1st 18-point encoder
	13	bit 0 - category 0, two-state 58
	14	1 - category 0, two-state 59
	15	2 - category 1, two-state 58
	16	3 - category 1, two-state 59
	17	4 - site category, two-state unassigned
	18	5 - site category, two-state 13
7	P0xxxxxx	1J5 (strapped for 3 bytes) - byte 1 of 2nd 18-point encoder
	19	bit 0 - site category, two-state 14
	20	1 - site category, two-state unassigned
	21	2 - site category, two-state 15
	22	3 - site category, two-state unassigned
	23	4 - site category, two-state unassigned
	24	5 - site category, two-state unassigned
8	P0xxxxxx	1J5 - byte 2 of 2nd 18-point encoder
	25	bit 0 - category 0, two-state 52
	26	1 - category 0, two-state 53
	27	2 - category 1, two-state 52
	28	3 - category 1, two-state 53
	29	4 - category 2, two-state 52
	30	5 - category 2, two-state 53
9	P0xxxxxx	1J5 - byte 3 of 2nd 18-point encoder
	31	bit 0 - category 2, two-state 54
	32	1 - category 2, two-state 55
	33	2 - category 1, two-state 54
	34	3 - category 1, two-state 55
	35	4 - category 2, two-state 54
	36	5 - category 2, two-state 55
10	P0xxxxxx	1J6 (strapped for 2 bytes) - byte 1 of 3rd 18-point encoder
	37	bit 0 - site category, two-state 16
	38	1 - site category, two-state 17
	39	2 - site category, two-state 18

```
   40           3 - site category, two-state unassigned
   41           4 - site category, two-state unassigned
   42           5 - site category, two-state unassigned
11  P0xxxxxx  1J6 - byte 2 of 3rd 18-point encoder
   43       bit 0 - site category, two-state unassigned
   44           1 - site category, two-state unassigned
   45           2 - site category, two-state unassigned
   46           3 - site category, two-state unassigned
   47           4 - site category, two-state unassigned
   48           5 - site category, two-state unassigned
12  P0xxxxxx  1J7 - byte 1 of 1st 12-point encoder
   49       bit 0 - site category, two-state 0
   50           1 - site category, two-state 1
   51           2 - site category, two-state 2
   52           3 - site category, two-state 3
   53           4 - site category, two-state 4
   54           5 - site category, two-state 5
13  P0xxxxxx  1J7 - byte 2 of 1st 12-point encoder
   55       bit 0 - site category, two-state 6
   56           1 - site category, two-state 7
   57           2 - site category, two-state 8
   58           3 - site category, two-state 9
   59           4 - site category, two-state 10
   60           5 - site category, two-state 11
14  P0xxxxxx  1J8 - byte 1 of 2nd 12-point encoder
   61       bit 0 - category 0, two-state 1
   62           1 - category 0, two-state 3
   63           2 - category 0, two-state 5
   64           3 - category 0, two-state 2
   65           4 - category 0, two-state 4
   66           5 - category 0, two-state 6
15  P0xxxxxx  1J8 - byte 2 of 2nd 12-point encoder
   67       bit 0 - category 0, two-state 7
   68           1 - category 0, two-state 0
   69           2 - category 0, two-state 8
   70           3 - category 0, two-state unassigned
   71           4 - category 0, two-state unassigned
   72           5 - category 0, two-state unassigned
16  P0xxxxxx  1J9 - byte 1 of 3rd 12-point encoder
   73       bit 0 - category 0, two-state 18
   74           1 - category 0, two-state 19
   75           2 - category 0, two-state 24
   76           3 - category 0, two-state 25
   77           4 - category 0, two-state 20
   78           5 - category 0, two-state 21
17  P0xxxxxx  1J9 - byte 2 of 3rd 12-point encoder
   79       bit 0 - category 0, two-state 22
   80           1 - category 0, two-state 23
   81           2 - site category, two-state unassigned
   82           3 - site category, two-state unassigned
   83           4 - site category, two-state unassigned
   84           5 - site category, two-state unassigned
```

```
18   P0xxxxxx  1J10 - byte 1 of 4th 12-point encoder
   85          bit 0 - category 1, two-state 1
   86              1 - category 1, two-state 3
   87              2 - category 1, two-state 5
   88              3 - category 1, two-state 2
   89              4 - category 1, two-state 4
   90              5 - category 1, two-state 6
19   P0xxxxxx  1J10 - byte 2 of 4th 12-point encoder
   91          bit 0 - category 1, two-state 7
   92              1 - category 1, two-state 0
   93              2 - category 1, two-state 8
   94              3 - category 1, two-state unassigned
   95              4 - category 1, two-state unassigned
   96              5 - category 1, two-state unassigned
20   P0xxxxxx  1J11 - byte 1 of 5th 12-point encoder
   97          bit 0 - category 1, two-state 18
   98              1 - category 1, two-state 19
   99              2 - category 1, two-state 24
  100              3 - category 1, two-state 25
  101              4 - category 1, two-state 20
  102              5 - category 1, two-state 21
21   P0xxxxxx  1J11 - byte 2 of 5th 12-point encoder
  103          bit 0 - category 1, two-state 22
  104              1 - category 1, two-state 23
  105              2 - site category, two-state unassigned
  106              3 - site category, two-state unassigned
  107              4 - site category, two-state unassigned
  108              5 - site category, two-state unassigned
22   P0xxxxxx  1J12 - byte 1 of 6th 12-point encoder
  109          bit 0 - category 2, two-state 1
  110              1 - category 2, two-state 3
  111              2 - category 2, two-state 5
  112              3 - category 2, two-state 2
  113              4 - category 2, two-state 4
  114              5 - category 2, two-state 6
23   P0xxxxxx  1J12 - byte 2 of 6th 12-point encoder
  115          bit 0 - category 2, two-state 7
  116              1 - category 2, two-state 0
  117              2 - category 2, two-state 8
  118              3 - category 2, two-state unassigned
  119              4 - category 2, two-state unassigned
  120              5 - category 2, two-state unassigned
24   P0xxxxxx  1J13 - byte 1 of 7th 12-point encoder
  121          bit 0 - category 2, two-state 18
  122              1 - category 2, two-state 19
  123              2 - category 2, two-state 24
  124              3 - category 2, two-state 25
  125              4 - category 2, two-state 20
  126              5 - category 2, two-state 21
25   P0xxxxxx  1J13 - byte 2 of 7th 12-point encoder
  127          bit 0 - category 2, two-state 22
  128              1 - category 2, two-state 23
```

129		2 - site category, two-state unassigned
130		3 - site category, two-state unassigned
131		4 - site category, two-state unassigned
132		5 - site category, two-state unassigned
26	P0xxxxxx	1J14 - byte 1 of 8th 12-point encoder
133		bit 0 - category 0, two-state 36
134		1 - category 0, two-state 37
135		2 - category 0, two-state 38
136		3 - category 0, two-state 39
137		4 - category 0, two-state 40
138		5 - category 0, two-state 41
27	P0xxxxxx	1J14 - byte 2 of 8th 12-point encoder
139		bit 0 - category 0, two-state 42
140		1 - category 0, two-state 43
141		2 - category 0, two-state 44
142		3 - category 0, two-state 45
143		4 - category 0, two-state 46
144		5 - category 0, two-state 47
28	P0xxxxxx	1J15 - byte 1 of 9th 12-point encoder
145		bit 0 - site category, two-state unassigned
146		1 - site category, two-state unassigned
147		2 - site category, two-state unassigned
148		3 - site category, two-state unassigned
149		4 - site category, two-state unassigned
150		5 - site category, two-state unassigned
29	P0xxxxxx	1J15 - byte 2 of 9th 12-point encoder
151		bit 0 - site category, two-state unassigned
152		1 - site category, two-state unassigned
153		2 - site category, two-state unassigned
154		3 - site category, two-state unassigned
155		4 - site category, two-state unassigned
156		5 - site category, two-state unassigned
30	01100011	Group ID character

Analog-to-Digital Parameters (5 bytes per card)

31	10000001	2J4, pin 17 - 1st Analog Value, **Parameter 0, category 0**
32	00000000	Scale ID (set to 0)
33	P00THHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
34	P000TTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
35	P000UUUU	Overflow bit(bit 4) and Units digit (BCD)
36	10000010	2J5, pin 17 - 2nd Analog Value, **Parameter 1, category 0**
37	00000000	Scale ID (set to 0)
38	P00THHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
39	P000TTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
40	P000UUUU	Overflow bit(bit 4) and Units digit (BCD)
41	00000011	2J6, pin 17 - 3rd Analog Value, **Parameter 2, category 0**
42	00000000	Scale ID (set to 0)
43	P00THHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
44	P000TTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
45	P000UUUU	Overflow bit(bit 4) and Units digit (BCD)

46	10000100	2J7, pin 17 - 1st Analog Value, **Parameter 0, category** 1
47	00000000	Scale ID (set to 0)
48	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
49	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
50	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
51	00000101	2J8, pin 17 - 2nd Analog Value, **Parameter 1, category** 1
52	00000000	Scale ID (set to 0)
53	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
54	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
55	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
56	00000110	2J9, pin 17 - 3rd Analog Value, **Parameter 2, category** 1
57	00000000	Scale ID (set to 0)
58	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
59	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
60	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
61	10000111	2J10, pin 17 - 1st Analog Value, **Parameter 0, category** 2
62	00000000	Scale ID (set to 0)
63	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
64	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
65	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
66	10001000	2J11, pin 17 - 2nd Analog Value, **Parameter 1, category** 2
67	00000000	Scale ID (set to 0)
68	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
69	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
70	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)
71	00001001	2J12, pin 17 - 3rd Analog Value, **Parameter 2, category** 2
72	00000000	Scale ID (set to 0)
73	POOTHHHH	Thousands digit (bit 4) and Hundreds digit (BCD)
74	POOOTTTT	Sign bit (bit 4, set to 0) and Tens digit (BCD)
75	POOOUUUU	Overflow bit(bit 4) and Units digit (BCD)

Digital Parameters (FEC Cards, 5 bytes per card)

76	01100011	2J17,Group/Point ID, Strap 13 = A, Switch S3 set to 143 Octal
77	P0xxxxxx	Parameter 0, Category 0, low-order 6 bits
78	P0xxxxxx	Parameter 0, Category 0, high-order 6 bits
79	P0xxxxxx	Parameter 1, Category 0, low-order 6 bits
80	P0xxxxxx	Parameter 1, Category 0, high-order 6 bits
81	11100100	2J18,Group/Point ID, Strap 13 = A, Switch S3 set to 144 Octal
82	P0xxxxxx	Parameter 0, Category 1, low-order 6 bits
83	P0xxxxxx	Parameter 0, Category 1, high-order 6 bits
84	P0xxxxxx	Parameter 1, Category 1, low-order 6 bits
85	P0xxxxxx	Parameter 1, Category 1, high-order 6 bits
86	01100101	3J3, Group/Point ID, Strap 13 = A, Switch S3 set to 145 Octal
87	P0xxxxxx	Parameter 0, Category 2, low-order 6 bits
88	P0xxxxxx	Parameter 0, Category 2, high-order 6 bits
89	P0xxxxxx	Parameter 1, Category 2, low-order 6 bits
90	P0xxxxxx	Parameter 1, Category 2, high-order 6 bits
91	11111111	End of Transmission character (ASCII <DELETE>)

Datalok 10 Poll and Response Character Formats

A typical character sent or received by a Datalok 10 Remote Unit has a start bit, 7 data bits, a parity bit, and 2 stop bits:

```
SXXXXXXXPTT  Where S = start bit (always = 0)
 01234567          X = data bit
 lo -  hi          P = parity bit
                   T = stop bit (always = 1)
```

Station ID characters, X = programmable data bit

```
XXXX001   1st character
XXXXA01   2nd character
              A = 1 for change of state (from Remote Unit)
                = 1 for manual interrogate (from Master)
XXXXA01   3rd character
              A = 0 for single master (from Master)
                = 1 for multi-master (from Master)
                = 1 (from Remote Unit)
```

Special characters, X = data bits

XXXXX11	Group ID
1111111	End of Transmission, ASCII <DELETE> character
XXXXXXX	Data character
XXXXXX0	Alarm Data character

Analog Data

AAAABB0	A = point ID, B = card ID
AABCCC0	A = card ID (continued), B = data valid, C = option code
XXXXAB0	X = 100's digit, A = 1/2 (1000's) digit, B = mode of multiplexer
XXXXA00	X = 10's digit, A = polarity (1 = positive, 0 = negative)
XXXXA00	X = 1's digit, A = overflow (1 = over-range)

FEC Card Data, FEC data is formatted in reverse order from other data (i.e. bit 5 first down to bit 0 last)

XXXXXX0	Error seconds (low order six bits)
XXXXXX0	Error seconds (high order six bits)
XXXXXX0	Error count (low order six bits)
XXXXXX0	Error count (high order six bits)

Relay (Switch) Commands. A switch command is made up of two characters, a card select and one of four relay commands.

XXXX011	Card Select
XXXX000	Relay interrogate (X = relay number)
XXXX010	Relay Preselect
XXXX100	Relay execute
UUUU110	Relay clear (clears all preselects), U = doesn't matter

Relay (Switch) Replies. A Switch reply is made up of two characters, a card reply and a relay reply.

XXXX111	Card reply
XXXXAB0	Relay reply(A 1 = energized, 0 = de-energized, B 1 = preselected)

APPENDIX F: DATALOK10, MODELS 1D AND 1E, DATA POINT ASSIGNMENTS

DATALOK10 Model 1E

DATALOK10 1E - Alarm/Status Assignments

Equip Rec'd Ord	Site			Linkend 0			Linkend 1		Linkend 2	
	T	Card	Pin	T	Card	Pin	Card	Pin	Card	Pin
0	L	1J8	1	L	1J9	1	1J12	1	1J15	1
1	L		2	L		2		2		2
2	L		3	L		3		3		3
3	L		4	L		4		4		4
4	L		5	L		5		5		5
5	L		6	L		6		6		6
6	L		7	L		7		7		7
7	L		8	L		8		8		8
8	L		9	L		9		9		9
9	L		10	L		10		10		10
10	L		11	L		11		11		11
11	L		12	L		12		12		12
12	M	1J7	1	L	1J10	1	1J13	1	1J16	1
13	M		2	L		2		2		2
14	M		3	L		3		3		3
15	M		4	L		4		4		4
16	M		5	L		5		5		5
17	M		6	L		6		6		6
18	M		7	L		7		7		7
19	M		8	L		8		8		8
20	M		9	L		9		9		9
21	M		10	L		10		10		10
22	M		11	L		11		11		11
23	M		12	L		12		12		12
24	M		13	L	1J11	1	1J14	1	1J17	1
25	M		14	L		2		2		2
26	M		15	L		3		3		3
27	M		16	L		4		4		4
28	M		17	L		5		5		5
29	M		18	L		6		6		6
30				L		7		7		7
31				L		8		8		8
32				L		9		9		9
33				L		10		10		10
34				L		11		11		11
35				L		12		12		12
36				L	1J18	1	2J2	5	2J3	9
37				L		2		6		10
38				L		3		7		11

DATALOK10 1E - Alarm/Status Assignments (cont.)

Equip Rec'd Ord	Site			Linkend 0			Linkend 1		Linkend 2	
	T	Card	Pin	T	Card	Pin	Card	Pin	Card	Pin
39				L	1J18	4	2J2	8	2J3	12
40				L		5		9	2J4	1
41				L		6		10		2
42				L		7		11		3
43				L		8		12		4
44				L		9	2J3	1		5
45				L		10		2		6
46				L		11		3		7
47				L		12		4		8
48				L	2J2	1		5		9
49				L		2		6		10
50				L		3		7		11
51				L		4		8		12
52				M	1J4	1	1J5	1	1J6	1
53				M		2		2		2
54				M		3		3		3
55				M		4		4		4
56				M		5		5		5
57				M		6		6		6
58				M		7		7		7
59				M		8		8		8
60				M		9		9		9
61				M		10		10		10
62				M		11		11		11
63				M		12		12		12
64				M		13		13		13
65				M		14		14		14
66				M		15		15		15
67				M		16		16		16
68				M		17		17		17
69				M		18		18		18

DATALOK10 1E - Analog Parameter Assignments

Equip Rec'd Ord	Site		Linkend 0		Linkend 1		Linkend 2	
	Card	Pin	Card	Pin	Card	Pin	Card	Pin
0	2J5	7	2J5	1	2J5	3	2J5	5
1		8		2		4		6
2		15		9		11		13
3		16		10		12		14

DATALOK10 1E - Digital Parameter Assignments

Equip Rec'd Ord	Site		Linkend 0		Linkend 1		Linkend 2	
	Card	Pin	Card	Pin	Card	Pin	Card	Pin
0			2J13	1	2J17	1	3J5	1
1								
2			2J14	1	2J18	1	3J6	1
3								
4			2J15	1	3J3	1	3J7	1
5								
6			2J16	1	3J4	1	3J8	1
7								

DATALOK10 1E - Switch Assignments

Equip Rec'd Ord	Site					Linkend 0					Linkend 1				Linkend 2			
	T	TB	Pins	CD	Rly	T	TB	Pins	CD	Rly	TB	Pins	CD	Rly	TB	Pins	CD	Rly
0	M	4B	13-15	1	5	M	4A	1-3	0	1	4B	19-21	1	7	5B	7-9	3	3
1	M		16-18	1	6	M		4-6	0	2		22-24	1	8		10-12	3	4
2	M	5B	1-3	3	1	M		7-9	0	3		25-27	1	9		13-15	3	5
3	M		4-6	3	2	M		10-12	0	4		28-30	1	10		16-18	3	6
4	M	6A	19-21	4	7	M		13-15	0	5	5A	1-3	2	1		19-21	3	7
5	M		22-24	4	8	M		16-18	0	6		4-6	2	2		22-24	3	8
6	M		25-27	4	9	M		19-21	0	7		7-9	2	3		25-27	3	9
7	M		28-30	4	10	M		22-24	0	8		10-12	2	4		28-30	3	10
8	L	6B	1-3	5	1	M		25-27	0	9		13-15	2	5	6A	1-3	4	1
9	L		22-24	6	3	M		28-30	0	10		16-18	2	6		4-6	4	2
10	L		25-27	6	4	M	4B	1-3	1	1		19-21	2	7		7-9	4	3
11	L		28-30	6	5	M		4-6	1	2		22-24	2	8		10-12	4	4
12						M		7-9	1	3		25-27	2	9		13-15	4	5
13						M		10-12	1	4		28-30	2	10		16-18	4	6
14						L	6B	4-6	5	2	6B	10-12	5	4	6B	16-18	6	1
15						L		7-9	5	3		13-15	5	5		19-21	6	2

DATALOK10 Model 1D

DATALOK10 1D - Alarm/Status Assignments

Equip Rec'd	Site			Linkend 0			Linkend 1		Linkend 2	
Ord	T	Card	Pin	T	Card	Pin	Card	Pin	Card	Pin
0	L	1J7	1	L	1J8	8	1J10	8	1J12	8
1	L		2	L		1		1		1
2	L		3	L		2		2		2
3	L		4	L		3		3		3
4	L		5	L		4		4		4
5	L		6	L		5		5		5
6	L		7	L		6		6		6
7	L		8	L		7		7		7
8	L		9	L		9		9		9
9	L		10	L		10		10		10
10	L		11	L		11		11		11
11	L		12	L		12		12		12
12										
13	M	1J4	18							
14	M	1J5	1							
15	M		3							
16	M	1J6	1							
17	M		2							
18	M		3	L	1J9	1	1J11	1	1J13	1
19				L		2		2		2
20				L		5		5		5
21				L		6		6		6
22				L		7		7		7
23				L		8		8		8
24				L		3		3		3
25				L		4		4		4
26				L		9		9		9
27				L		10		10		10
28				L		11		11		11
29				L		12		12		12
30										
31										
32										
33										
34										
35										
36				L	1J14	1	1J14	9	1J15	5
37				L		2		10		6
38				L		3		11		7

DATALOK10 1D - Alarm/Status Assignments (cont.)

Equip Rec'd Ord	Site			Linkend 0			Linkend 1		Linkend 2	
	T	Card	Pin	T	Card	Pin	Card	Pin	Card	Pin
39				L	1J14	4	1J14	12	1J15	8
40				L		5	1J15	1	2J4	9
41				L		6		2		10
42				L		7		3		11
43				L		8		4		12
44										
45										
46										
47										
48										
49										
50										
51										
52				M	1J5	7	1J5	9	1J5	11
53				M		8		10		12
54				M		13		15		17
55				M		14		16		18
56				M	1J4	7	1J4	9	1J4	11
57				M		8		10		12
58				M		13		15		
59				M		14		16		
60										
61										
62										
63										
64										
65										
66										
67										
68										
69										

DATALOK10 1D - Analog Parameter Assignments

Equip Rec'd Ord	Site		Linkend 0		Linkend 1		Linkend 2	
	Card	Pin	Card	Pin	Card	Pin	Card	Pin
0			2J4	17	2J7	17	2J10	17
1			2J5	17	2J8	17	2J11	17
2			2J6	17	2J9	17	2J12	17

DATALOK10 1D - Digital Parameter Assignments

Equip Rec'd Ord	Site		Linkend 0		Linkend 1		Linkend 2	
	Card	Pin	Card	Pin	Card	Pin	Card	Pin
0			2J13	1	2J17	1	3J5	1
1								
2			2J14	1	2J18	1	3J6	1
3								
4			2J15	1	3J3	1	3J7	1
5								
6			2J16	1	3J4	1	3J8	1
7								

DATALOK10 1D - Switch Assignments

Equip Rec'd Ord	Site					Linkend 0					Linkend 1				Linkend 2			
	T	TB	Pin	CD	Rly	T	TB	Pin	CD	Rly	TB	Pin	CD	Rly	TB	Pin	CD	Rly
0	L	3B	1-3	2	1													
1	L		4-6		2	M	4B	7- 9	4	3	4B	13-15	4	5	4B	19-21	4	7
2	L		7-9		3	M		10-12		4		16-18		6		22-24		8
3	L		13-15		5	L	3A	19-21	1	2	3A	25-27	1	4				
4	M	4B	25-27	4	9	L		22-24		3		28-30		5				
5																		
6																		
7																		
8																		
9																		
10						M	4A	4- 6	3	2	4A	16-18	3	6	4A	28-30	3	0
11						M		1- 3		1		13-15		5		25-27		9
12						M		10-12		4		22-24		8	4B	4- 6	4	2
13						M		7- 9		3		19-21		7		1- 3		1
14						L	3A	1- 3	0	1	3A	7- 9	0	3	3A	13-15	0	5
15						L		4- 6		2		10-12		4		16-18	1	1

APPENDIX G: OPTIMIZING THE LOADING OF SEGMENTED PROGRAMS

This appendix illustrates how the loading of the large segmented programs can be modifed by hand to produce a more efficient program and dramatically improve the resource requirements of these programs.

Specifically, these modifications can reduce the number of segments and, therefore, reduce swapping activity and improve the execution of the large program. By placing the redundent modules into the root node, the size of the type 6 program file can be reduced. For the memory-resident segmented programs, MTRP and PLRP, moving the redundent modules to the root node also results in a significant reduction in the memory partition space required. This is important because memory is already scarce.

The modification is performed in two stages. In Step one, the loader directive file that is produced by the segmenter program SGMTR is examined for modules that are specified to be loaded in every path of the program. These NA and SY directives should be moved to the root node. After the first step, the program should be loaded and the load map should be stored on a disc file. Step two examines the load map for routines that were not mentioned explicitly in the loader directive file, but are still being loaded into every path. Explicit NA or SY directives for these modules should be added to the loader directive file in the root node.

The example shown here is the optimization of the memory-resident program MTRP. Before modification, program MTRP requires a dedicated 49 page memory partition and the size of the type 6 disc file is 266 blocks. Listed below is the loader directive file #MTRP for program MTRP before modifcation.

```
  * RU,SGMTR,@MTRP,#XX::10,28,MTRP,M  *  2:55 PM  MON., 22  FEB., 1988
SH,SHAR1
SZ,30
LI,@MTRP
LI,$PLDH2
OP,EM
OP,BP
* *TOTAL PROGRAM SIZE IN DECIMAL     34322  *SGMTR:      3 NODES CREATED
M
NA,MTRP
NA,PAS.1
NA,PAS.2
NA,PAS.STOP
NA,PAS.CURRMARC2
NA,PAS.GETMEMINFO2
NA,PAS.HIWATERHEAP2
NA,PAS.HIWATERMARK2
NA,PAS.LOWATERHEAP2
NA,PAS.LOWATERMARK2
NA,PAS.PREVFREE2
NA,PAS.SETMEMINFO2
NA,PAS.TOPOFHEAP2
NA,PAS.TOPOFSTACK2
```

```
NA,PAS.GETNEWPARMS
NA,PAS.INITIALIZE
NA,PAS.NUMERICPARMS
NA,PAS.RUNSTRINGLEN
NA,PAS.RUNSTRINGPTR
NA,PAS.STRENDS
NA,PAS.BITMASK0
NA,PAS.BITMASK1
SY,RMPAR
SY,D$XFR
SY,D.R
SY,R/W$
SY,.OFLG
SY,RFLG$
SY,RWND$
SY,WFLG$
SY,.DBTS
SY,.BFSZ
SY,$EMA$
SY,$SWP$
SY,L$PTE
SY,S$PTE
SY,VMAST
SY,$LOC$
SY,$LOD$
SY,.RRGR
SY,.SVRG
SY,OVRD.
M.1
NA,UPDATE_CURSOR
NA,GET_ANSWER
NA,UPDATE_DISPLAYS
NA,SET_DATA_FRAME
NA,PAS.CLOSEFILE
NA,PAS.CDSCONFLICT
NA,PAS.NONCDS
NA,PAS.TRACEBEGIN
NA,PAS.TRACECLOSE
NA,PAS.TRACEEND
NA,PAS.TRACEINIT
NA,PAS.INITMEMINFO2
NA,PAS.INITFILE
M.2
NA,PM_INIT
NA,PROCESS_RESPONSE
NA,UPDATE_US
NA,SET_DATA_FRAME
NA,PAS.CLOSEFILE
NA,PAS.CDSCONFLICT
NA,PAS.NONCDS
END
```

For step one, the four underlined NA directives (SET_DATA_FRAME, PAS.CLOSEFILE, PAS.CDSCONFLICT, AND PAS.NONCDS) appearing in all paths in the listing above are moved to the root node. The program is reloaded and the load map is stored on a disc file. The resulting load map is listed below. Usually, but not always, there is at least one routine (of the same name as in the NA directive) appearing in the load map for every NA directive in the loader input file. For example, the NA directive NA,SET_DATA_FRAME results in the loading of routine SET_DATA_FRAME as shown in the second line of node 0 below. As an exception, the two NA directives NA,CDSCONFLICT AND NA,NONCDS result in just the one line PAS.NONCDSLIB in node 0 below.

```
M
NODE       0

MTRP                2012  7252  Monitor Response Handler      890420.1020
SET_DATA_FRAME      7253  7354  TRAMCON Library, Ver. DEV     890328.0916
PAS.MEMDATA2        7355  7423  92833-16119 REV.2440 841008
PAS.INITIALIZE      7424  7522  92833-16119 REV.2440 841008
PAS.BITMASK         7523  7564  92833-16119 REV.2440 841008
PAS.CLOSEFILE       7565 10222  92833-16108,REV.2440,850215   850215.1702
PAS.NONCDSLIB      10223 10222  92833-16119 REV.2440 841008
PAS.DCBADDRESS1    10223 10255  92833-16119 REV.2440 841008
PAS.FILEERROR      10256 10310  92833-16112,REV.2440,850215   850215.1659
PAS.UPSHIFTALPHA   10311 10352  92833-16118,REV.2440,850215   850215.1648
PAS.WRITELINE      10353 10524  92833-16108,REV.2440,850215   850215.1702
PAS.ERRORCATCHER   10525 10610  92833-16112,REV.2440,850215   850215.1659
PAS.IOERROR        10611 10643  92833-16112,REV.2440,850215   850215.1659
PAS.ERRORPRINTER   10644 13674  92833-16112,REV.2440,850215   850215.1659
PAS.TRACEBACK      13675 13701  92833-16119 REV.2440 841008
PAS.BOUNDINTEGER   13702 13733  92833-16119 REV.2440 841008
PAS.DOUBLE2ASCII   13734 14133  92833-16118,REV.2440,850215   850215.1648
PAS.STRINGADDRS    14134 14141  92833-16119 REV.2440 841008

$EMA$              14142 14301  92084-1X085 REV.2440 <841114.1544>
$LOC$              14302 14353  92084-1X415 REV.2121 810723
RMPAR              14354 14404  92084-1X069 REV.2121 811001
R/W$               14405 14522  92077-1X532 REV.2340 830217
.OFLG              14523 14525  92084-1Y010 REV.2340 830628
RWND$              14526 14660  92077-1X534 REV.2326 <830217.1306>
.DBTS              14661 14664  92084-1Y010 REV.2340 830628
.BFSZ              14665 14761  92077-1X658 REV.2340 830226
$SWP$              14762 15151  92084-1X086 REV.2440 <841114.1528>
L$PTE              15152 15172  92084-1X099 REV.2121 801204
VMAST              15173 15241  92084-1X101 REV.2121 810513
.RRGR              15242 15276  92084-1X419 REV.2121 810723
OVRD.              15277 15277  92077-1X482 REV.2340 830218
CLOSE              15300 15530  92077-1X536 REV.2340 830819
LOCF               15531 16046  92077-1X539 REV.2326 <830217.1307>
RWNDF              16047 16137  92077-1X529 REV.2326 <830217.1306>
READF              16140 17531  92077-1X528 REV.2440 <840730.0942>
LOGLU              17532 17607  92084-1X027 REV.2121 790228
PRTN               17610 17722  92084-1X007 REV.2121 771005
```

```
  XREIO            17723 20073  92084-1X923 REV.2440 840228
  SYSRQ            20074 20167  92084-1X002 REV.2121 810831
  RW$UB            20170 20562  92077-1X533 REV.2326 <830217.1306>

  M.1
  NODE      1

  GET_ANSWER       22011 24661  TRAMCON MPLIB, Ver. DEV        890421.1125
  PAS.INITFILE     24662 24726  92833-16119 REV.2440 841008
  PAS.INITMEMINFO2 24727 25006  92833-16119 REV.2440 841008
  PAS.TRACEDUMMY   25007 25011  92833-16119 REV.2440 841008
  UPDATE_CURSOR    25012 26323  TRAMCON MPLIB, Ver. DEV        890421.1125
  UPDATE_DISPLAYS  26324 31556  TRAMCON MPLIB, Ver. DEV        890421.1125
ELAPSEDTIME      31557 31733  TRAMCON Library, Ver. DEV      890328.0916
PAS.BLANKFILL    31734 32032  92833-16119 REV.2440 841008
PAS.REWRITE_FILE 32033 32147  92833-16118,REV.2440,850215    850215.1648
PAS.WRITECHAR    32150 32232  92833-16118,REV.2440,850215    850215.1648
  PAS.WRITEENUM    32233 32522  92833-16118,REV.2440,850215    850215.1648
PAS.WRITEINTEGER 32523 32552  92833-16118,REV.2440,850215    850215.1648
PAS.WRITESTRING  32553 32776  92833-16118,REV.2440,850215    850215.1648
  PRINT_RESPONSE   32777 37374  TRAMCON MPLIB, Ver. DEV        890421.1125
  PAS.INITMEMINFO1 37375 37437  92833-16119 REV.2440 841008
PAS.RUNTIMEERROR 37440 37461  92833-16112,REV.2440,850215    850215.1659
  CLEAR_CHARS      37462 40016  TRAMCON MPLIB, Ver. DEV        890421.1125
DISABLE_KEYBOARD 40017 40214  TRAMCON Library, Ver. DEV      890328.0916
KEYPRESS         40215 41664  TRAMCON Library, Ver. DEV      890328.0916
  ARCHIVE_IT       41665 43004  TRAMCON MPLIB, Ver. DEV        890421.1125
  GET_SITE_STATUS  43005 43773  TRAMCON Library, Ver. DEV      890328.0916
  JTIME            43774 44257  TRAMCON Library, Ver. DEV      890328.0916
  READ_DICT        44260 44426  TRAMCON Library, Ver. DEV      890328.0916
  RING_AUDIBLE     44427 44610  TRAMCON Library, Ver. DEV      890328.0916
  UPDATE_AL        44611 47147  TRAMCON MPLIB, Ver. DEV        890421.1125
  UPDATE_CN        47150 50140  TRAMCON MPLIB, Ver. DEV        890421.1125
  UPDATE_PA        50141 50662  TRAMCON MPLIB, Ver. DEV        890421.1125
  UPDATE_PC        50663 51447  TRAMCON MPLIB, Ver. DEV        890421.1125
  UPDATE_SS        51450 53246  TRAMCON MPLIB, Ver. DEV        890421.1125
  PAS.SETUPFILE    53247 55673  92833-16108,REV.2440,850215    850215.1702
PAS.IOWARNING    55674 55726  92833-16112,REV.2440,850215    850215.1659
PAS.PUT          55727 56235  92833-16108,REV.2440,850215    850215.1702
PAS.MOVEBYTES    56236 56265  92833-16119 REV.2440 841008
PAS.SPLITMOVE    56266 56514  92833-16118,REV.2440,850215    850215.1648
PAS.WRITEDOUBLE  56515 56676  92833-16118,REV.2440,850215    850215.1648
  PAS.WRITEREAL    56677 56747  92833-16118,REV.2440,850215    850215.1648
REVERSE_BITS     56750 57064  TRAMCON MPLIB, Ver. DEV        890421.1125
  PAS.MEMDATA1     57065 57122  92833-16119 REV.2440 841008
CRT_STATUS_CHECK 57123 57175  TRAMCON Library, Ver. DEV      890328.0916
PAS.PROMPT       57176 57304  92833-16108,REV.2440,850215    850215.1702
  PAS.BITOPERATOR2 57305 57765  92833-16119 REV.2440 841008
PAS.OPEN_FILE    57766 60044  92833-16118,REV.2440,850215    850215.1648
  PAS.SEEKFILE     60045 60145  92833-16118,REV.2440,850215    850215.1648
  PAS.WRITENONTEXT 60146 60220  92833-16118,REV.2440,850215    850215.1648
  PRINT_VAL        60221 61155  TRAMCON MPLIB, Ver. DEV        890421.1125
```

```
PAS.CLOSEPURGE    61156 61174  92833-16118,REV.2440,850215     850215.1648
PAS.INLINEERROR   61175 61206  92833-16119 REV.2440 841008
PAS.SINGLEMOD     61207 61242  92833-16119 REV.2440 841008
  PAS.WRITEANYREAL 61243 61444  92833-16118,REV.2440,850215     850215.1648
DOWN_CRT          61445 61740  TRAMCON Library, Ver. DEV       890328.0916
  PARM_DEF          61741 62074  TRAMCON MPLIB, Ver. DEV          890421.1125
  PRINT_PARM        62075 62175  TRAMCON MPLIB, Ver. DEV          890421.1125
  PAS.REAL2ASCII    62176 63404  92833-16118,REV.2440,850215      850215.1648
  PAS.MAX           63405 63442  92833-16118,REV.2440,850215      850215.1648
  PAS.MIN           63443 63500  92833-16118,REV.2440,850215      850215.1648
  PAS.REALOPERATOR 63501 64703  92833-16119 REV.2440 841008
  PAS.LOADRHEAP2    64704 64703  92833-16117 REV.2401 840322

ABREG             64704 64725  92084-1X059 REV.2121 750701
IAND              64726 64735  24998-1X102 REV.2001 750701
  LIMEM             64736 64777  92084-1X050 REV.2121 810717
  CNUMD             65000 65017  92084-1X015 REV.2121 770621
  KCVT              65020 65033  92084-1X011 REV.2121 770621
CREAT             65034 65420  92077-1X379 REV.2326 <830218.1103>
NAMR              65421 65720  92084-1X066 REV.2226 820225
OPENF             65721 66227  92077-1X541 REV.2440 841012
POST              66230 66257  92077-1X527 REV.2326 <830217.1319>
.DMOD             66260 66277  24998-1X269 REV.2101 800303
MESSS             66300 66640  92084-1X458 REV.2440 <841005.1346>
  .LWAS             66641 66641  92084-1X411 REV.2121 810717
  COR.A             66642 66662  92084-1X009 REV.2121 770621
  $CVT3             66663 66750  92084-1X018 REV.2121 770621
$OPEN             66751 67201  92077-1X544 REV.2326 830905
NAM..             67202 67276  92077-1X530 REV.2340 830217
OPEN              67277 67710  92077-1X088 REV.2440 841011
IFTTY             67711 70000  92084-1X025 REV.2301 820916
$ESTB             70001 70015  92084-1X048 REV.2121 790202
CAPCK             70016 70366  92084-1X028 REV.2121 810126
IDGET             70367 70451  92084-1X029 REV.2121 790314
VSCBA             70452 70521  92084-1X461 REV.2121 810201
$SMVE             70522 70614  92084-1X046 REV.2121 800129
SESSN             70615 70632  92084-1X256 REV.2121 780413

  M.2
  NODE       2

  PM_INIT           22011 22670  Monitor Response Handler         890420.1020
  PROCESS_RESPONSE 22671 31507  TRAMCON MPLIB, Ver. DEV          890421.1125
  UPDATE_US         31510 33160  TRAMCON MPLIB, Ver. DEV          890421.1125
  ALLOCATE_EMA      33161 34015  TRAMCON Library, Ver. DEV        890328.0916
  PAS.BITOPERATOR2 34016 34476  92833-16119 REV.2440 841008
PAS.OPEN_FILE     34477 34555  92833-16118,REV.2440,850215     850215.1648
ELAPSEDTIME       34556 34732  TRAMCON Library, Ver. DEV       890328.0916
  EVALUATE_NODE     34733 35266  TRAMCON MPLIB, Ver. DEV          890421.1125
  PARM_DEF          35267 35422  TRAMCON MPLIB, Ver. DEV          890421.1125
  PAS.REALROUND     35423 35475  92833-16118,REV.2440,850215      850215.1648
  PAS.SETDIFFER     35476 35527  92833-16119 REV.2440 841008
  PAS.SETINIT       35530 35661   92833-16119 REV.2440 841008
PAS.WRITEINTEGER 35662 35711   92833-16118,REV.2440,850215     850215.1648
PAS.WRITESTRING  35712 36135   92833-16118,REV.2440,850215     850215.1648
  UNPACK_RESPONSE  36136 40011   TRAMCON MPLIB, Ver. DEV          890421.1125
DISABLE_KEYBOARD 40012 40207   TRAMCON Library, Ver. DEV       890328.0916
KEYPRESS         40210 41657   TRAMCON Library, Ver. DEV       890328.0916
PAS.SINGLEMOD    41660 41713   92833-16119 REV.2440 841008
PAS.WRITECHAR    41714 41776   92833-16118,REV.2440,850215     850215.1648
PAS.WRITEDOUBLE  41777 42160   92833-16118,REV.2440,850215     850215.1648
PAS.REWRITE_FILE 42161 42275   92833-16118,REV.2440,850215     850215.1648
  PAS.SETSHARED    42276 42444   92833-16118,REV.2440,850215     850215.1648
  PAS.SETUPFILE    42445 45071   92833-16108,REV.2440,850215     850215.1702
  PAS.ENTRYEXIT2   45072 45646   92833-16119 REV.2440 841008
PAS.RUNTIMEERROR 45647 45670   92833-16112,REV.2440,850215     850215.1659
PAS.MOVEBYTES    45671 45720   92833-16119 REV.2440 841008
PAS.PUT          45721 46227   92833-16108,REV.2440,850215     850215.1702
PAS.SPLITMOVE    46230 46456   92833-16118,REV.2440,850215     850215.1648
REVERSE_BITS     46457 46573   TRAMCON MPLIB, Ver. DEV         890421.1125
  TRANSFORM_ORDINA 46574 50266   TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.BLANKFILL    50267 50365   92833-16119 REV.2440 841008
CRT_STATUS_CHECK 50366 50440   TRAMCON Library, Ver. DEV       890328.0916
PAS.PROMPT       50441 50547   92833-16108,REV.2440,850215     850215.1702
PAS.IOWARNING    50550 50602   92833-16112,REV.2440,850215     850215.1659
  PAS.SHAREDSIZE   50603 50707   92833-16118,REV.2440,850215     850215.1648
PAS.CLOSEPURGE   50710 50726   92833-16118,REV.2440,850215     850215.1648
PAS.INLINEERROR  50727 50740   92833-16119 REV.2440 841008
  PAS.CHECKSTAKSZ2 50741 50770   92833-16119 REV.2440 841008
DOWN_CRT         50771 51264   TRAMCON Library, Ver. DEV       890328.0916

.DMOD            51265 51304   24998-1X269 REV.2101 800303
IAND             51305 51314   24998-1X102 REV.2001 750701
ABREG            51315 51336   92084-1X059 REV.2121 750701
CREAT            51337 51723   92077-1X379 REV.2326 <830218.1103>
NAMR             51724 52223   92084-1X066 REV.2226 820225
OPENF            52224 52532   92077-1X541 REV.2440 841012
POST             52533 52562   92077-1X527 REV.2326 <830217.1319>
  IXGET            52563 52572   92084-1X030 REV.2121 780731
MESSS            52573 53133   92084-1X458 REV.2440 <841005.1346>
$OPEN            53134 53364   92077-1X544 REV.2326 830905
NAM..            53365 53461   92077-1X530 REV.2340 830217
OPEN             53462 54073   92077-1X088 REV.2440 841011
IFTTY            54074 54163   92084-1X025 REV.2301 820916
$ESTB            54164 54200   92084-1X048 REV.2121 790202
CAPCK            54201 54551   92084-1X028 REV.2121 810126
IDGET            54552 54634   92084-1X029 REV.2121 790314
VSCBA            54635 54704   92084-1X461 REV.2121 810201
$SMVE            54705 54777   92084-1X046 REV.2121 800129
SESSN            55000 55015   92084-1X256 REV.2121 780413

      29 PAGE LONGEST PATH
      45 PAGE PARTITION REQUIRED
```

Already, the partition size has been reduced from 49 pages to 45 pages. This frees four pages that can be used to increase the size of another program partition or to increase the size of the shared memory EMA partition. Also, the type 6 disc file has been reduced from 385 blocks to 353 blocks. Step two identifies all the modules (underlined in the listing above) that are still loaded into every path (the example has two paths - M1 and M2). These modules are moved to the root node resulting in the loader directive file listed below. Notice that not all underlined routines are explicitly specified with NA or SY directives in the new loader directive file. For example, most of the system library routines, such as $OPEN and SESSN, do not have a corresponding SY directive. These routines do not require explicit specification because they are loaded in the proper node automatically by the loader which loads all modules referenced by the explicitly specified modules. That is, when loading the module PAS.OPENFILE, the loader realizes that it must also load the module $OPEN since it is called by the module PAS.OPENFILE. To keep the loader directive file manageable, care should be taken to explicitly specify only those modules that need to be specified. This list of necessary modules can be determined by trial and error. Explicitly mention those modules that you think need to be mentioned and attempt to load the program. If the loader pauses looking for undefined externals, then those modules must also be mentioned in an NA or SY directive.

```
  * RU,SGMTR,@MTRP,#XX::10,28,MTRP,M  *  2:55 PM  MON., 22  FEB., 1988
SH,SHAR1
SZ,30
LI,@MTRP
LI,$PLDH2
OP,EM
OP,BP
* *TOTAL PROGRAM SIZE IN DECIMAL          34322  *SGMTR:      3 NODES CREATED
M
NA,MTRP
NA,SET_DATA_FRAME
NA,KEYPRESS
NA,DISABLE_KEYBOARD
NA,ELAPSEDTIME
NA,REVERSE_BITS
NA,PAS.1
NA,PAS.2
NA,PAS.STOP
NA,PAS.CURRMARC2
NA,PAS.GETMEMINFO2
NA,PAS.HIWATERHEAP2
NA,PAS.HIWATERMARK2
NA,PAS.LOWATERHEAP2
NA,PAS.LOWATERMARK2
NA,PAS.PREVFREE2
NA,PAS.SETMEMINFO2
NA,PAS.TOPOFHEAP2
NA,PAS.TOPOFSTACK2
NA,PAS.GETNEWPARMS
```

```
NA,PAS.INITIALIZE
NA,PAS.NUMERICPARMS
NA,PAS.RUNSTRINGLEN
NA,PAS.RUNSTRINGPTR
NA,PAS.STRENDS
NA,PAS.BITMASK0
NA,PAS.BITMASK1
NA,PAS.CLOSEFILE
NA,PAS.CDSCONFLICT
NA,PAS.NONCDS
NA,PAS.OPEN_FILE
NA,PAS.WRITEINTEGER
NA,PAS.WRITESTRING
NA,PAS.SINGLEMOD
NA,PAS.SETUPFILE
NA,PAS.RUNTIMEERROR
NA,PAS.WRITEDOUBLE
NA,PAS.MOVEBYTES
NA,PAS.PUT
NA,PAS.SPLITMOVE
NA,PAS.CLOSEPURGE
NA,PAS.INLINEERROR
NA,PAS.IOWARNING
SY,RMPAR
SY,D$XFR
SY,D.R
SY,R/W$
SY,.OFLG
SY,RFLG$
SY,RWND$
SY,WFLG$
SY,.DBTS
SY,.BFSZ
SY,$EMA$
SY,$SWP$
SY,L$PTE
SY,S$PTE
SY,VMAST
SY,$LOC$
SY,$LOD$
SY,.RRGR
SY,.SVRG
SY,OVRD.
SY,.DMOD
M.1
NA,UPDATE_CURSOR
NA,GET_ANSWER
NA,UPDATE_DISPLAYS
NA,PAS.TRACEBEGIN
NA,PAS.TRACECLOSE
NA,PAS.TRACEEND
NA,PAS.TRACEINIT
```

```
NA,PAS.INITMEMINFO2
NA,PAS.INITFILE
M.2
NA,PM_INIT
NA,PROCESS_RESPONSE
NA,UPDATE_US
END
```

Again, the program is reloaded and the resulting load map is listed below.

```
M
NODE       0

MTRP               2012  7252  Monitor Response Handler        890420.1020
SET_DATA_FRAME     7253  7354  TRAMCON Library, Ver. DEV       890328.0916
KEYPRESS           7355 11024  TRAMCON Library, Ver. DEV       890328.0916
DISABLE_KEYBOARD  11025 11222  TRAMCON Library, Ver. DEV       890328.0916
ELAPSEDTIME       11223 11377  TRAMCON Library, Ver. DEV       890328.0916
REVERSE_BITS      11400 11514  TRAMCON MPLIB, Ver. DEV         890421.1125
PAS.MEMDATA2      11515 11563  92833-16119 REV.2440 841008
PAS.INITIALIZE    11564 11662  92833-16119 REV.2440 841008
PAS.BITMASK       11663 11724  92833-16119 REV.2440 841008
PAS.CLOSEFILE     11725 12362  92833-16108,REV.2440,850215     850215.1702
PAS.NONCDSLIB     12363 12362  92833-16119 REV.2440 841008
PAS.OPEN_FILE     12363 12441  92833-16118,REV.2440,850215     850215.1648
PAS.WRITEINTEGER  12442 12471  92833-16118,REV.2440,850215     850215.1648
PAS.WRITESTRING   12472 12715  92833-16118,REV.2440,850215     850215.1648
PAS.SINGLEMOD     12716 12751  92833-16119 REV.2440 841008
PAS.SETUPFILE     12752 15376  92833-16108,REV.2440,850215     850215.1702
PAS.RUNTIMEERROR  15377 15420  92833-16112,REV.2440,850215     850215.1659
PAS.WRITEDOUBLE   15421 15602  92833-16118,REV.2440,850215     850215.1648
PAS.MOVEBYTES     15603 15632  92833-16119 REV.2440 841008
PAS.PUT           15633 16141  92833-16108,REV.2440,850215     850215.1702
PAS.SPLITMOVE     16142 16370  92833-16118,REV.2440,850215     850215.1648
PAS.CLOSEPURGE    16371 16407  92833-16118,REV.2440,850215     850215.1648
PAS.INLINEERROR   16410 16421  92833-16119 REV.2440 841008
PAS.IOWARNING     16422 16454  92833-16112,REV.2440,850215     850215.1659
CRT_STATUS_CHECK  16455 16527  TRAMCON Library, Ver. DEV       890328.0916
PAS.BLANKFILL     16530 16626  92833-16119 REV.2440 841008
PAS.PROMPT        16627 16735  92833-16108,REV.2440,850215     850215.1702
PAS.WRITECHAR     16736 17020  92833-16118,REV.2440,850215     850215.1648
PAS.REWRITE_FILE  17021 17135  92833-16118,REV.2440,850215     850215.1648
PAS.DCBADDRESS1   17136 17170  92833-16119 REV.2440 841008
PAS.FILEERROR     17171 17223  92833-16112,REV.2440,850215     850215.1659
PAS.UPSHIFTALPHA  17224 17265  92833-16118,REV.2440,850215     850215.1648
PAS.WRITELINE     17266 17437  92833-16108,REV.2440,850215     850215.1702
PAS.IOERROR       17440 17472  92833-16112,REV.2440,850215     850215.1659
PAS.ERRORCATCHER  17473 17556  92833-16112,REV.2440,850215     850215.1659
PAS.DOUBLE2ASCII  17557 17756  92833-16118,REV.2440,850215     850215.1648
DOWN_CRT          17757 20252  TRAMCON Library, Ver. DEV       890328.0916
PAS.ERRORPRINTER  20253 23303  92833-16112,REV.2440,850215     850215.1659
PAS.TRACEBACK     23304 23310  92833-16119 REV.2440 841008
PAS.BOUNDINTEGER 23311 23342  92833-16119 REV.2440 841008
PAS.STRINGADDRS  23343 23350  92833-16119 REV.2440 841008

$EMA$            23351 23510  92084-1X085 REV.2440 <841114.1544>
$LOC$            23511 23562  92084-1X415 REV.2121 810723
RMPAR            23563 23613  92084-1X069 REV.2121 811001
R/W$             23614 23731  92077-1X532 REV.2340 830217
.OFLG            23732 23734  92084-1Y010 REV.2340 830628
RWND$            23735 24067  92077-1X534 REV.2326 <830217.1306>
.DBTS            24070 24073  92084-1Y010 REV.2340 830628
.BFSZ            24074 24170  92077-1X658 REV.2340 830226
$SWP$            24171 24360  92084-1X086 REV.2440 <841114.1528>
L$PTE            24361 24401  92084-1X099 REV.2121 801204
VMAST            24402 24450  92084-1X101 REV.2121 810513
.RRGR            24451 24505  92084-1X419 REV.2121 810723
OVRD.            24506 24506  92077-1X482 REV.2340 830218
.DMOD            24507 24526  24998-1X269 REV.2101 800303
ABREG            24527 24550  92084-1X059 REV.2121 750701
IAND             24551 24560  24998-1X102 REV.2001 750701
SYSRQ            24561 24654  92084-1X002 REV.2121 810831
XREIO            24655 25025  92084-1X923 REV.2440 840228
CLOSE            25026 25256  92077-1X536 REV.2340 830819
LOCF             25257 25574  92077-1X539 REV.2326 <830217.1307>
RWNDF            25575 25665  92077-1X529 REV.2326 <830217.1306>
CREAT            25666 26252  92077-1X379 REV.2326 <830218.1103>
READF            26253 27644  92077-1X528 REV.2440 <840730.0942>
LOGLU            27645 27722  92084-1X027 REV.2121 790228
NAMR             27723 30222  92084-1X066 REV.2226 820225
OPENF            30223 30531  92077-1X541 REV.2440 841012
POST             30532 30561  92077-1X527 REV.2326 <830217.1319>
PRTN             30562 30674  92084-1X007 REV.2121 771005
MESSS            30675 31235  92084-1X458 REV.2440 <841005.1346>
$OPEN            31236 31466  92077-1X544 REV.2326 830905
NAM..            31467 31563  92077-1X530 REV.2340 830217
RW$UB            31564 32156  92077-1X533 REV.2326 <830217.1306>
OPEN             32157 32570  92077-1X088 REV.2440 841011
IFTTY            32571 32660  92084-1X025 REV.2301 820916
$ESTB            32661 32675  92084-1X048 REV.2121 790202
CAPCK            32676 33246  92084-1X028 REV.2121 810126
IDGET            33247 33331  92084-1X029 REV.2121 790314
VSCBA            33332 33401  92084-1X461 REV.2121 810201
$SMVE            33402 33474  92084-1X046 REV.2121 800129
SESSN            33475 33512  92084-1X256 REV.2121 780413

M.1
NODE        1

GET_ANSWER       34011 36661  TRAMCON MPLIB, Ver. DEV         890421.1125
PAS.INITFILE     36662 36726  92833-16119 REV.2440 841008
PAS.INITMEMINFO2 36727 37006  92833-16119 REV.2440 841008
PAS.TRACEDUMMY   37007 37011  92833-16119 REV.2440 841008
UPDATE_CURSOR    37012 40323  TRAMCON MPLIB, Ver. DEV         890421.1125
```

```
UPDATE_DISPLAYS    40324 43556  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.WRITEENUM      43557 44046  92833-16118,REV.2440,850215      850215.1648
PRINT_RESPONSE     44047 50444  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.INITMEMINFO1   50445 50507  92833-16119 REV.2440 841008
CLEAR_CHARS        50510 51044  TRAMCON MPLIB, Ver. DEV          890421.1125
ARCHIVE_IT         51045 52164  TRAMCON MPLIB, Ver. DEV          890421.1125
GET_SITE_STATUS    52165 53153  TRAMCON Library, Ver. DEV        890328.0916
JTIME              53154 53437  TRAMCON Library, Ver. DEV        890328.0916
READ_DICT          53440 53606  TRAMCON Library, Ver. DEV        890328.0916
RING_AUDIBLE       53607 53770  TRAMCON Library, Ver. DEV        890328.0916
UPDATE_AL          53771 56327  TRAMCON MPLIB, Ver. DEV          890421.1125
UPDATE_CN          56330 57320  TRAMCON MPLIB, Ver. DEV          890421.1125
UPDATE_PA          57321 60042  TRAMCON MPLIB, Ver. DEV          890421.1125
UPDATE_PC          60043 60627  TRAMCON MPLIB, Ver. DEV          890421.1125
UPDATE_SS          60630 62426  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.WRITEREAL      62427 62477  92833-16118,REV.2440,850215      850215.1648
PAS.MEMDATA1       62500 62535  92833-16119 REV.2440 841008
PAS.BITOPERATOR2   62536 63216  92833-16119 REV.2440 841008
PAS.SEEKFILE       63217 63317  92833-16118,REV.2440,850215      850215.1648
PAS.WRITENONTEXT   63320 63372  92833-16118,REV.2440,850215      850215.1648
PRINT_VAL          63373 64327  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.WRITEANYREAL   64330 64531  92833-16118,REV.2440,850215      850215.1648
PARM_DEF           64532 64665  TRAMCON MPLIB, Ver. DEV          890421.1125
PRINT_PARM         64666 64766  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.REAL2ASCII     64767 66175  92833-16118,REV.2440,850215      850215.1648
PAS.MAX            66176 66233  92833-16118,REV.2440,850215      850215.1648
PAS.MIN            66234 66271  92833-16118,REV.2440,850215      850215.1648
PAS.REALOPERATOR   66272 67474  92833-16119 REV.2440 841008
PAS.LOADRHEAP2     67475 67474  92833-16117 REV.2401 840322

LIMEM              67475 67536  92084-1X050 REV.2121 810717
CNUMD              67537 67556  92084-1X015 REV.2121 770621
KCVT               67557 67572  92084-1X011 REV.2121 770621
.LWAS              67573 67573  92084-1X411 REV.2121 810717
COR.A              67574 67614  92084-1X009 REV.2121 770621
$CVT3              67615 67702  92084-1X018 REV.2121 770621

M.2
NODE        2

PM_INIT            34011 34670  Monitor Response Handler         890420.1020
PROCESS_RESPONSE   34671 43507  TRAMCON MPLIB, Ver. DEV          890421.1125
UPDATE_US          43510 45160  TRAMCON MPLIB, Ver. DEV          890421.1125
ALLOCATE_EMA       45161 46015  TRAMCON Library, Ver. DEV        890328.0916
PAS.BITOPERATOR2   46016 46476  92833-16119 REV.2440 841008
EVALUATE_NODE      46477 47032  TRAMCON MPLIB, Ver. DEV          890421.1125
PARM_DEF           47033 47166  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.REALROUND      47167 47241  92833-16118,REV.2440,850215      850215.1648
PAS.SETDIFFER      47242 47273  92833-16119 REV.2440 841008
PAS.SETINIT        47274 47425  92833-16119 REV.2440 841008
UNPACK_RESPONSE    47426 51301  TRAMCON MPLIB, Ver. DEV          890421.1125
PAS.SETSHARED      51302 51450  92833-16118,REV.2440,850215      850215.1648
```

```
  PAS.ENTRYEXIT2     51451 52225  92833-16119 REV.2440 841008
  TRANSFORM_ORDINA 52226 53720  TRAMCON MPLIB, Ver. DEV         890421.1125
  PAS.SHAREDSIZE     53721 54025  92833-16118,REV.2440,850215    850215.1648
  PAS.CHECKSTAKSZ2 54026 54055  92833-16119 REV.2440 841008

  IXGET              54056 54065  92084-1X030 REV.2121 780731

      28 PAGE LONGEST PATH
      40 PAGE PARTITION REQUIRED
```

These two simple optimization steps have reduced program MTRPs' dedicated partition requirements from 49 pages to 40 pages and reduced the type 6 file size from 385 disc blocks to 305 blocks. This same process could be applied to program PLRP resulting in a total memory savings of 18 pages and disc space savings of 160 blocks.

INDEX

Page

FORM **NTIA-29**
(4-80)

U.S. DEPARTMENT OF COMMERCE
NAT'L. TELECOMMUNICATIONS AND INFORMATION ADMINISTRATION

BIBLIOGRAPHIC DATA SHEET

	1. PUBLICATION NO.	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE
Transmission Monitor and Control Software Reference Manual

5. Publication Date
June 1990

6. Performing Organization Code
NTIA/ITS

7. AUTHOR(S)
Richard N. Statz

9. Project/Task/Work Unit No.

8. PERFORMING ORGANIZATION NAME AND ADDRESS
National Telecommunications & Information Administration
Institute for Telecommunication Sciences
325 Broadway
Boulder, CO 80303

10. Contract/Grant No.

11. Sponsoring Organization Name and Address
U.S. Air Force
Electronics Systems Division
Hanscom AFB, MA 01731

12. Type of Report and Period Covered

13.

14. SUPPLEMENTARY NOTES

15. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

This manual describes the functions of the TRAMCON (TRAnsmission Monitor and CONtrol) On-Line software and the steps necessary to maintain this software. This document emphasizes the software semantics rather than the syntax. The structure of the software is described and the design, and development strategies used in the creation of the software is explained. This manual is intended to provide assistance to experienced programers who want to change or enhance the TRAMCON On-Line software.

16. Key Words *(Alphabetical order, separated by semicolons)*
automated technical control; data archiving; Digital European Backbone; microwave radio; network management; polling; pulse count; software; TRAMCON; transmission monitor and control.

17. AVAILABILITY STATEMENT	18. Security Class. *(This report)* Unclassified	20. Number of pages 307
[✓] UNLIMITED. ☐ FOR OFFICIAL DISTRIBUTION.	19. Security Class. *(This page)* Unclassified	21. Price: